IMPLEMENTING COOPERATIVE BEHAVIOR & CONTROL USING OPEN SOURCE TECHNOLOGY ACROSS HETEROGENEOUS VEHICLES

THESIS
MARCH 2015

Stefan L. Hardy, 1st Lieutenant, USAF

AFIT-ENV-MS-15-M-180

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-MS-15-M-180

IMPLEMENTING COOPERATIVE BEHAVIOR & CONTROL USING OPEN SOURCE TECHNOLOGY ACROSS HETEROGENOUS VEHICLES

THESIS

Presented to the Faculty

Department of Systems Engineering and Management

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Systems Engineering

Stefan L. Hardy, BS

1st Lieutenant, USAF

March 2015

AFIT-ENV-MS-15-M-180

IMPLEMENTING COOPERATIVE BEHAVIOR & CONTROL USING OPEN SOURCE TECHNOLOGY ACROSS HETEROGENEOUS VEHICLES

Stefan L. Hardy, BS

1st Lieutenant, USAF

Committee Membership:

Dr. David R. Jacques
Chair

Dr. John M. Colombi
Member

Maj. Brian B. Stone, Ph.D.
Member

Abstract

This thesis describes the research effort into implementing cooperative behavior and control across heterogeneous vehicles using low cost off-the-shelf technologies and open source software. Current cooperative behavior and control methods are explored and improved upon to build analysis models. These analysis models characterize ideal factor settings for implementation and establish limits of performance for these low cost approaches to cooperative behavior and control.

The research focused on latency and position accuracy as the two measures of performance. Three different ground control station (GCS) software applications and two types of vehicles, rover ground vehicles and aerial multi-rotors, were used in this research. Using optimum factor settings from Design of Experiments (DOE), the multi-rotor following rover vehicle configuration experienced almost twice the latency of other experiments but also the lowest positional error of 0.8 m. Results show that the achieved update frequency of 0.5 Hz or slower would be far too slow for close-formation flight.

Acknowledgments

I would like to thank my chairman, Dr. Jacques, and my committee members, Dr. Colombi and Maj Stone, for their unlimited support. Dr. Jacques was in the trenches with me throughout my research, from extended 2+ hour long discussions/meetings, to freezing alongside me during experimental testing.

This document is a testament to my wife, who has given about as much as I have, if not more, for me to be where I am today. Her insurmountable support has carried me through the doubtful moments, inspiring confidence in me in moments of reservation. I've learned more from her than any research can provide. We had our first child right before the first quarter began. Though daunting, you made raising a newborn and working towards a master's degree feel effortless. She's given me the joy of being a husband and a father, everything I've ever wanted.

God is the reason for everything I am and everything I have. He has never led me astray. I owe everything I have to Him. For that, I am blessed.

Stefan L. Hardy

Table of Contents

Page

Abstract iv
Table of Contents vi
List of Figures ix
List of Tables xiii
I. Introduction 1
Background 1
Problem Statement 2
Research Objectives/Questions 3
Methodology 5
Assumptions/Limitations 6
Implications 7
Preview 8
II. Literature Review 9
Chapter Overview 9
Policy 9
Cooperative Behavior 10
Cooperative Control 12
Metrics 19
Summary 25
III. Methodology 27
Chapter Overview 27
Materials and Equipment 27
Configuration Architectures 35

Procedures 45
Summary.....57
IV. Analysis and Results.....60
Chapter Overview.....60
Diagnostic Testing.....60
Optimum Factor Settings (Design of Experiments) 63
Heterogeneous Vehicle Implementation 88
Summary.....94
V. Application Analysis.....96
Chapter Overview.....96
Close-Formation Flight 96
Target Information Sharing 97
Vehicle Following 97
Investigative Questions 103
VI. Conclusions and Recommendations 108
Chapter Overview.....108
Conclusions of Research 108
Significance of Research 109
Recommendations for Action.....112
Recommendations for Future Research.....114
Bibliography 116
Appendix A: Traxxas Modified Rover Ground Vehicle Setup 120
Appendix B: X8 Multi-Rotor Setup.....126
Appendix C: Leader Vehicle Python Script.....132
Appendix D: Rover Follower Distance Offset Python Script.....134

Appendix E: Rover Follower Vehicle Heading Offset Python Script 137
Appendix F: Multi-Rotor Follower Vehicle Heading Offset Python Script.....140
Appendix G: AFIT Document 5028 Test Project Technical and Safety Review 144

List of Figures

Page

Figure 1. Heterogeneous Vehicles In Operation [1] 2

Figure 2. Mission Planner Swarm Application With Grid Offset [3] 14

Figure 3. Flocking Algorithm Run Through Python [13] 15

Figure 4. 3DR Pixhawk Autopilot [21] 29

Figure 5. 915 MHz 3DRobotics Telemetry Modems [23] 30

Figure 6. Rover Ground Vehicles 31

Figure 7. X8 Multi-Rotor [25] 32

Figure 8. Mission Planner 33

Figure 9. Droid Planner 2 [27] 35

Figure 10. Mission Planner Swarm SV-1 (Configuration 1) 36

Figure 11. Python Method on One GCS SV-1 (Configuration 2) 37

Figure 12. Python Method on Separate GCSs SV-1 (Configuration 3) 38

Figure 13. Droid Planner 2 Method SV-1 (Configuration 4) 39

Figure 14. Vehicle System Node Functions SV-4 40

Figure 15. Mission Planner System Node Functions SV-4 41

Figure 16. Python System Node Functions SV-4 42

Figure 17. Droid Planner 2 System Node Functions SV-4 44

Figure 18. Operator System Node Functions SV-4 45

Figure 19. Latency Model Screening 64

Figure 20. Latency Half Normal Plot 64

Figure 21. Latency Effect Tests 65

Figure 22. Latency Summary of Fit 66
Figure 23. Latency ANOVA Table 66
Figure 24. Latency Residual vs. Predicted Plot 66
Figure 25. Latency Normality Plot 67
Figure 26. Latency Lack of Fit Test 67
Figure 27. Latency Parameter Estimates 68
Figure 28. Latency Lack of Fit Test Without Sleep Time^2 In Model 68
Figure 29. Latency Summary of Fit Without Sleep Time^2 In Model 68
Figure 30. Latency Prediction Profiler 70
Figure 31. Latency Cube Plot 71
Figure 32. Latency Interaction Plots 72
Figure 33. Accuracy Error Summary of Fit 74
Figure 34. Accuracy Error ANOVA Table 74
Figure 35. Accuracy Error Effect Tests 74
Figure 36. Accuracy Error Lack of Fit Test 75
Figure 37. Accuracy Error Parameter Estimates 75
Figure 38. Accuracy Error Prediction Profiler 76
Figure 39. Accuracy Error Cube Plot 76
Figure 40. Accuracy Error Residual vs. Predicted Plot 77
Figure 41. Accuracy Error Normality Plot 77
Figure 42. Accuracy Error Box-Cox Transformation 78
Figure 43. Square Root Accuracy Error Transformation Summary of Fit 78
Figure 44. Square Root Accuracy Error Transformation Parameter Estimates 78

Figure 45. Square Root Accuracy Error Transformation Residual vs. Predicted Plot 79
Figure 46. Log Accuracy Error Transformation Summary of Fit..... 79
Figure 47. Log Accuracy Error Transformation Parameter Estimates 79
Figure 48. Log Accuracy Error Transformation Residual vs. Predicted Plot..... 80
Figure 49. Figure Eight Accuracy Error Model Screening..... 81
Figure 50. Figure Eight Accuracy Error Half Normal Plot 81
Figure 51. Figure Eight Accuracy Error Effect Tests..... 82
Figure 52. Figure Eight Accuracy Error Summary of Fit..... 82
Figure 53. Figure Eight Accuracy Error ANOVA Table..... 82
Figure 54. Figure Eight Accuracy Error Parameter Estimates 83
Figure 55. Figure Eight Accuracy Error Prediction Profiler..... 83
Figure 56. Figure Eight Accuracy Error Cube Plots..... 84
Figure 57. Figure Eight Accuracy Error Interaction Plots..... 85
Figure 58. Figure Eight Accuracy Error Residual vs. Predicted Plot 86
Figure 59. Figure Eight Accuracy Error Normality Plot 86
Figure 60. Figure Eight Accuracy Error Box-Cox Transformation..... 86
Figure 61. Sideview of Camera Footprint..... 99
Figure 62. Bird's Eye View of Camera Footprint 101
Figure 63. Rover Gain Settings..... 120
Figure 64. Rover Steering Modes 122
Figure 65. Rover Components 122
Figure 66. Traxxas Rover Battery..... 123
Figure 67. X8 Multi-Rotor Gain Settings 127

Figure 68. X8 Multi-Rotor Flight Modes 129

Figure 69. X8 Multi-Rotor Components..... 129

Figure 70. Multi-Rotor Battery..... 130

List of Tables

Page

Table 1. Clough's Autonomous Control Level (ACL) Chart [17] 20

Table 2. Unmanned Vehicle Human Supervisory Control Metric Classes and Subclasses [18] 21

Table 3. Dudek's Taxonomy Properties Used In The Experiments [20] 24

Table 4. DOE Factor Levels 53

Table 5. Test 2 Five Factor Half-Fractional Factorial Design 55

Table 6. Diagnostic Testing for MP Swarm and Python Configurations 61

Table 7. Two-Sample T-Test Results Between Python Configurations 62

Table 8. Optimal Factor Settings for Low Latency 73

Table 9. Optimal Factor Settings for Low Accuracy Error 80

Table 10. Optimal Factor Settings for Low Figure Eight Accuracy Error 87

Table 11. Heterogeneous Vehicle Implementation 89

Table 12. Multi-Rotor Following Rover Droid Planner 2 Tests 89

Table 13. Commanded Offset vs. Actual Distance Accuracy 93

Table 14. Commanded Offset vs. Actual Distance Accuracy CEP 94

Table 15. Angle Calculations for Azimuth FOV 99

Table 16. Footprint Distances 100

Table 17. Rover Travel Time/Latency Buffers 102

Table 18. Rover Gain Settings 121

Table 19. Rover Components 124

Table 20. MXL-6s ESC Speed Controller Specifications 125

Table 21. X8 Multi-Rotor Gain Settings 128

Table 22. X8 Multi-Rotor Components 131

IMPLEMENTING COOPERATIVE BEHAVIOR & CONTROL USING OPEN SOURCE TECHNOLOGY ACROSS HETEROGENEOUS VEHICLES

I. Introduction

This thesis describes the research effort into implementing cooperative behavior and control across heterogeneous vehicles using off the shelf technologies and open source software. Current cooperative behavior and control methods are explored and improved upon to build analysis models. These analysis models characterize ideal factor settings for use in heterogeneous vehicle implementation and establish limits of performance for these low cost approaches to cooperative behavior and control.

Background

Heterogeneous vehicles are defined as using a combination of non-similar vehicles such as rover ground vehicles, multi-rotors, planes, and other vehicles with different capabilities. For this research, heterogeneous vehicles will be assumed to be low cost, ranging from a couple of hundred dollars to a couple of thousand dollars. These low cost vehicles are more expendable than current expensive UAVs, allowing for even riskier missions without fear of no return, and flexible designs small enough to be used by ground troops on the frontlines. The vehicles communicate with a Ground Control Station (GCS) for control. Figure 1 below is a visual of rovers, multi-rotors, and planes, operating together in theatre [1]. This particular opeation shows the payload drops of rover vehicles from a plane, while the multi-rotors are launched from off of the rovers to prep for surveillance missions.

Figure 1. Heterogeneous Vehicles In Operation [1]

Problem Statement

Low cost heterogeneous vehicles can use cooperative behavior and control to support applications in the military. Still, because of their affordability, these heterogeneous vehicles could be used against the United States military by its adversaries. Though the limits of performance are unknown, it's important to identify the limitations of these low cost heterogeneous vehicles using cooperative behavior and control. Identifying these limitations will aid in recognizing suitable applications for the United States military and for preparation in deterring adversary use. Some ways that cooperative behavior and control can benefit the military is through addressing communication and fragile vehicle issues.

Communications are currently limited with vehicles in urban warfare or other restricted communication environments, and mission effectiveness is currently limited by the robustness of the vehicle. The robustness of the vehicle could be measured by the ability to respond to the operator's controls, ability to process autonomous functions such as distributed waypoints, or even the ability to physically withstand the operational terrain or environment. The DoD is currently looking for methods to improve the robustness of UAVs.

These issues are reason to incorporate cooperative behavior and control into multiple vehicles, but not necessarily of the same type. Heterogeneity offers the ability to address another specific set of problems. In some instances, the environment can be the most crucial issue or weakness among autonomous vehicles. The operational environment can vary in many ways such as urban, rural, mountains, plains, land, sea, dry, or wet. Therefore, incorporating multiple vehicles of the same type to complete a mission limits the vehicles to the same terrain. For instance, a rover does not have the same view as an aerial vehicle would. If the rover were to take a path that would later prove to be impossible to navigate through, the rover would lose valuable time and resources back-tracking to a more accessible path, or even worse, not be able to continue the mission at all. Aerial vehicles could be used as a scout to communicate with the GCS or the rover as to where the optimum route would be [2]. These aerial vehicles, such as the multi-rotors or planes, have a view that the ground rovers do not.

In enclosed areas or coverings, aerial vehicles could be arguably useless. In these circumstances, rovers could navigate through pipes, low lying coverings, or places where aerial vehicles would have a hard time operating. Therefore, by implementing cooperative behavior and control into heterogeneous vehicles, the weaknesses of each vehicle could be counteracted with the strengths of the accompanying vehicles.

Research Objectives/Questions

The primary question for this research is, given the state of technology for commercially, available autopilots and Remote Control (RC) hobbyist equipment, what is the achievable performance for cooperative behavior among heterogeneous vehicles? However, there are other hurdles to cross in order to answer this question.

First, cooperative behavior and control must be proven on a particular platform, whether it be rovers, multi-rotors, or planes. The GCS software, platform, and method should all be accounted for.

A baseline can be established in order to improve cooperative behavior and control. A new method of cooperative behavior and control can then be started, noting the software, vehicles, and method being used. This new method can offer integration of further cooperative behavior applications. The baseline method could then be used in comparison to the improved method of cooperative behavior and control. The data files from these experiment comparisons should then be analyzed to conclude the effects of these methods. With an improved method, performance can be optimized through experimentation to find ideal settings, which can be used on differing heterogeneous vehicle configurations.

Research Focus

There are many different approaches to, or hardware sets, that can be used with cooperative behavior and control applications. For example, expensive autopilots for vehicle navigation and autonomy offer high processing capabilities, but are not as accessible or disposable as low cost equipment. Therefore, this research is scoped to focus on low cost readily available technology.

Investigative Questions

In order to the research question several other questions need to be addressed:

1. What methods are currently available for cooperative behavior and control with low cost vehicles? Research will be done to discover current methods of cooperative behavior and control for low cost vehicles.
2. What are the challenges of using multiple heterogeneous vehicles from a single GCS? The limitations or challenges involved with implementing heterogeneous vehicles from a single GCS could limit the performance of cooperative behavior and control.

3. What is the initial architecture that can be implemented and improved upon? A baseline architecture will need to be established in order to improve cooperative behavior and control.
4. What appropriate assessment measures should be used for analysis? These assessment measures will be used to define the effectiveness of cooperative behavior and control throughout the research.
5. What are the performance limitations given current architecture? Once an improved architecture for cooperative behavior and control is established, experimentation may reveal limitations or shortcomings in performance.
6. What cooperative behavior applications are reasonable or achievable given current limitations? Potential cooperative behavior applications will be discussed given the results and limitations of the current cooperative behavior and control architecture.

Methodology

A baseline will need to be established for cooperative behavior and control for heterogeneous vehicles. Therefore, cooperative behavior and control will first need to be implemented on homogenous vehicles in order to verify implementation of GCS software applications. The GCS software applications will include a "swarm" capability that will set one vehicle as a leader and the others as followers [3]. This capability will pave the way for new cooperative behavior and control methods. Using a programming language, the same capabilities from the "swarm" function will be implemented across multiple GCS instances. By re-creating this application using scripting, it can be improved upon. Once again, homogenous vehicles will be used in the same tests as the GCS software application tests. Therefore, accuracy error and latency will be available for proper assessment and comparison.

After the baseline comparisons, the new programmed method will incorporate new behaviors. Using homogenous vehicles, experimentation and analysis software will help find the ideal settings for optimum performance. Performance measures will assess low latency and accuracy.

The newly improved programming script can then be implemented on two heterogeneous vehicles, a rover ground vehicle and a multi-rotor. The same ideal settings from homogenous vehicle tests can then be used on heterogeneous vehicles to assess the same latency and accuracy measures in order to establish comparative assessments. Another method, involving a smart phone application, will be used to measure latency and accuracy error in heterogeneous vehicle configurations as well, as an alternative method implementation.

The data collected will allow for analysis and calculations regarding mounted vehicle camera performance. These calculations will offer insight into how operating parameters and design choices will affect the camera's footprint from aerial vehicles, attempting to maintain surveillance over a ground vehicle.

Assumptions/Limitations

All tests will be done outdoors. Depending on when and where the tests are executed, the weather should be favorable, including dry, warm, calm weather. Once temperatures reach below freezing, the battery life on the vehicles and operating times for the safety pilot start to diminish rapidly making the mission time unpredictable. Windy weather will add too much noise to the data by pushing vehicles off course. Therefore, testing needed to be complete prior to the winter months.

Global Positioning System (GPS) reception will be necessary for leader/follower navigation and for any cooperative behavior related to navigation, relative to position. GPS and internet are required for the synchronization of maps on the GCS, which is used for waypoint

selection. GPS is the heart of autonomous navigation. Without GPS, the heart of low cost cooperative behavior and control dies alongside autonomous navigation. When operating vehicles on separate GCSs, a network connection between the GCSs must also be established.

Aerial vehicles may not be flown by the military without approval. Therefore, when planning to fly aerial vehicles, a proper location must be selected that supports UAV testing. Camp Atterbury, IN offers a testing range for UAVs with a devoted UAV runway and restricted military use airpsace. Appropriate accommodations must be made in order to reserve the testing site and use all scheduled times efficiently since the location is 145 miles away.

Upgrades to GCS software are constantly being released and the version of the software and firmware that are on the vehicles should be known at all times. If not, there can be compatibility issues and the vehicles may not respond to commands from the GCS.

Cooperative behavior and control has been an area that many have focused on improving. As noted earlier, open source GCS software recently developed a beta "swarm" application as an attempt to make strides in the cooperative behavior community. Many others have tried incorporating cooperative behavior and control into heterogeneous vehicles [2]. However, this research focuses on demonstrating what has already been established in open source software and using it as a way to customize another method of cooperative behavior and control for heterogeneous vehicles. This new cooperative behavior method will improve the baseline method with the integration of new behaviors and capabilities using programming scripts.

Implications

Successfully establishing an architecture for cooperative behavior and control for heterogeneous vehicles will give the DoD and other UAV or low cost vehicle users an edge in the field. Cooperative behavior and control could help improve communications by allowing each vehicle to act as a relay in a link of communications to the GCS, increasing the range of missions.

UAVs could be more robust with the addition of multiple vehicles flying in formation or heterogeneous vehicles used to overcome obstacles that single or homogenous vehicles couldn't do on their own. Using each vehicle's strength to counteract the other's weakness will allow for maximum mission effectiveness.

Alternating a leader in a group of heterogeneous vehicles adds a dimension of flexibility and allows for every vehicle to equally distribute resources. If one were to fail, several others could step up and take its place.

Preview

The subsequent chapters will present additional material on cooperative behavior and control for heterogeneous vehicles. Chapter two will discuss the literature involved with the research. It will discuss what cooperative behavior and control research has been done before, autonomy assessments for verification, and military flight policy for multiple vehicle operation.

Chapter three discuss the methodology of the research. It will explain what experiments are to be done, what data will be obtained, how they will be obtained, and how they will be analyzed. The chapter will discuss the details of the software and procedures used.

Chapter four will present the results of the research and the analysis associated with them. The data will be presented in the form of plots, spreadsheets, and algorithms.

Chapter five will cover application analysis from this research. A sequence of calculations and a trade study will investigate how certain variables should be altered to maximize the effectiveness of an application in a heterogeneous vehicle configuration.

Chapter six will disclose the conclusions associated with the research after analyzing the data. The chapter will offer final thoughts and explanations of the research, what was done, what could have been done better, and future areas of work for follow-on research.

II. Literature Review

Chapter Overview

The purpose of this chapter is to provide relevant background information and report on the investigations and results of other researchers. Terms will be defined, and a review of recent literature will validate the focus of this research. Research into current issues will validate the focus of this research. Previous research efforts will be presented that establish foundations for this research.

The focus of this research is to implement cooperative behavior across heterogeneous vehicles using off the shelf technology and open source software. This research involves addressing methods of cooperative behavior and control with heterogeneous vehicles that can be used and improved. Defense Advanced Research Projects Agency (DARPA) recently initiated a Swarm Challenge Program, with a goal to "leverage affordable, existing unmanned systems and platforms and/or low-cost approaches enabled by distributed/redundant functionality of heterogeneous unmanned systems," as one of its goals of a new program [1]. While the current research is not associated with the DARPA program, there is an overlap in the research objectives. In subsequent sections, policy will be discussed, cooperative behavior and control will be defined, and metrics used for evaluation will be identified.

Policy

There are guidelines, such as GCS configuration, altitude, and speed restrictions on SUAS, written in policy that must be followed, which could limit this research. Both the Federal Aviation Administration (FAA) and the Air Force have policy and guidance associated with small UAS flight, and both need to be understood.

There are five groups of UAS codified by the JFCOM Joint UAS CONOP, but only the first two will be discussed due to their relevance to this research [4]. Group 1 involves UAS less

than 20 pounds that normally operate below 1200 feet Above Ground Level (AGL) and at speeds less than 250 knots. Group 2 UAS weigh between 21 and 55 pounds and operate below 3500 feet AGL at less than 250 knots. All vehicles related to this research fall under Group 1 classification. It is important to note the classifications because there are specific restrictions related to each group classification. However, Group 1 UAS are exempt from most of the restrictions placed on the other groups. Air Force Institute of Technology's (AFIT) current Military Flight Release (MFR) prevents any military UAS from flying outside reserved training locations without a Certificate of Authority (COA) from the FAA. The closest facility to test UAS in military restricted airspace is Camp Atterbury, IN. A further restriction prevents multiple UAS from flying under control of a single GCS [5]. This is a form of policy that this research hopes to change after the groundwork is established for proof of air worthiness of cooperative behavior and control for heterogeneous vehicles. Until then, this research will focus on heterogeneous vehicles, consisting of rover ground vehicles and aerial multi-rotors. If more than one aerial vehicle will be used simultaneously, it will operate from separate GCSs, using a network connection between them. The policy and restrictions discussed are primarily the Department of Defense's (DOD) guidelines; FAA guidelines weren't discussed since no vehicle was flown outside of military restricted airspace for this research.

Cooperative Behavior

Cooperative behavior and control are two separate functions; however, sometimes they overlap. Cooperative behavior describes the act of cooperation between two or more vehicles, which can be facilitated by cooperative control. It is more formally defined as, "the interaction of two or more persons or organizations directed toward a common goal which is mutually beneficial. An act or instance of working or acting together for a common purpose or benefit" [6]. Cooperative behavior was originally learned from the behaviors of animals, insects, and

people. Therefore, the realm of cooperative behavior evolved from natural living organisms to autonomous vehicles, which may share some similar characteristics of biological system. Cooperative behavior brings another dimension of autonomy to man-made vehicles. Some say cooperative behavior can exist only when individuals improve the joint payoff, instead of their own payoff [7]. Thus, cooperation involves joining two or more vehicles to reach a more beneficial goal or task, that would not be achievable with individual performance.

Flocking/Swarming

Some types of cooperative behavior include swarming and flocking. Many people use these definitions interchangeably as a collective behavior of individuals interacting with one another towards a common direction [8]. Some use "flock" to describe a behavior and "swarm" to describe a group of individuals. In biological terms, swarming refers to the collective behavior of a group of insects, and flocking refers to the collective behavior of a group of birds. If the group of individuals were acting like a flock of birds, the rapid collective moving to and away from locations, then the term "flocking" can be used in the description [9]. If the group of individuals was acting like a swarm of insects, the constant collective movement of a group around a location, then one might use the term "swarming." This research will focus on the biological definitions of the two words in reference to vehicle behavior. In terms of planes, flocking may be more commonly used because it involves the non-hovering constant flight, mostly seen with birds. In terms of multi-rotors or rover ground vehicles, swarming may be more commonly used because it involves hovering capabilities, or a stop and go characteristic, that planes are not able to exhibit.

Impact

Cooperative behavior and control among unmanned vehicles can support mission capability by extending the range of communications. Low cost unmanned vehicles typically are limited to line-of-sight (LOS) communication range. Using cooperative behavior & control and

Commercial Off-The-Shelf (COTS) hardware, a rover and relay Small Unmanned Aerial System (SUAS were used to essentially double the communication range from the GCS to the rover aircraft [10]. The relay aircraft used an algorithm to effectively navigate to an optimal position for communication range based on the location and heading of the rover vehicle.

With the success that cooperative behavior and control brings to the mission, heterogeneity adds another dimension. Heterogeneity allows flexibility in a multi-terrain, unpredictable operational environment. For example, a multi-rotor SUAS was used in combination with multiple ground vehicles to overcome terrain obstacles [2]. A single ground vehicle was incapable of navigating over a steep ramp. However, with the ability for multiple ground vehicles to attach to one another, the vehicles could generate enough force to collectively maneuver over the ramp. The ground vehicles, using the multi-rotor vehicle's camera vision, used internal processing and programmed behaviors to interconnect. The multi-rotor vehicle was able to display a view beyond the vision of the ground vehicles, detecting a need for cooperative behavior. In this instance, the multi-rotor essentially acts as a scout. This scout configuration could be beneficial for a ground vehicle, whose camera doesn't have the range of vision of multi-rotors. Otherwise, the ground vehicle could unknowingly maneuver onto an obstacle it cannot navigate over due to its lack of preemptive vision.

Cooperative Control

Cooperative control is a precursor to cooperative behavior. It involves the control of individuals or vehicles to perform cooperative behavior. Cooperative control can include algorithms, feedback loops, and formation control [11]. Cooperative control essentially explains the "How," while cooperative behavior explains the "What." Cooperative control explains how the vehicles or individuals will interact and what measures will enforce it.

Swarm

Mission Planner, an open source GCS software managed by Michael Oborne, has a "swarm" application developed through contributions of the open source community [3]. This application is a beta feature and is continuously being updated. It allows for the connection of more than one autonomous vehicle, setting one vehicle as a leader and one or more as followers. The leader can maneuver by manual control or autonomously through given waypoints. The followers, given an offset, will follow the leader autonomously. The application offers the ability to set the leader-follower offset. A grid feature displays the location of the connected vehicles. By dragging each vehicle on the grid, the follower offset from the leader can be established. It should be noted that this offset is relative to inertial bearing. For example, if the follower is positioned directly behind the leader on the grid, the follower will follow directly behind the leader only when the leader travels North. If the leader were to maneuver East, the follower would be attempting to follow the leader in a parallel fashion instead of directly behind. The grid determines an offset based only on North, South, East, and West position or a geodetic frame. This research will preserve the follower's formation by configuring the offset relative to the leader heading instead of the leader's geodetic orientation. Nevertheless, the application's arguably best contribution is the ability to simultaneously connect to multiple vehicles. The application adds the ability to add a second Micro Air Vehicle Link (MAVLink), which is a link between the GCS and the vehicle that communicates the GPS location, speed, and other vehicle parameters. Mission Planner is not able to connect to multiple vehicles without the addition of the "swarm" application. Figure 2 below shows the swarm application menu with the grid offset on Mission Planner, and the second MAVLink option.

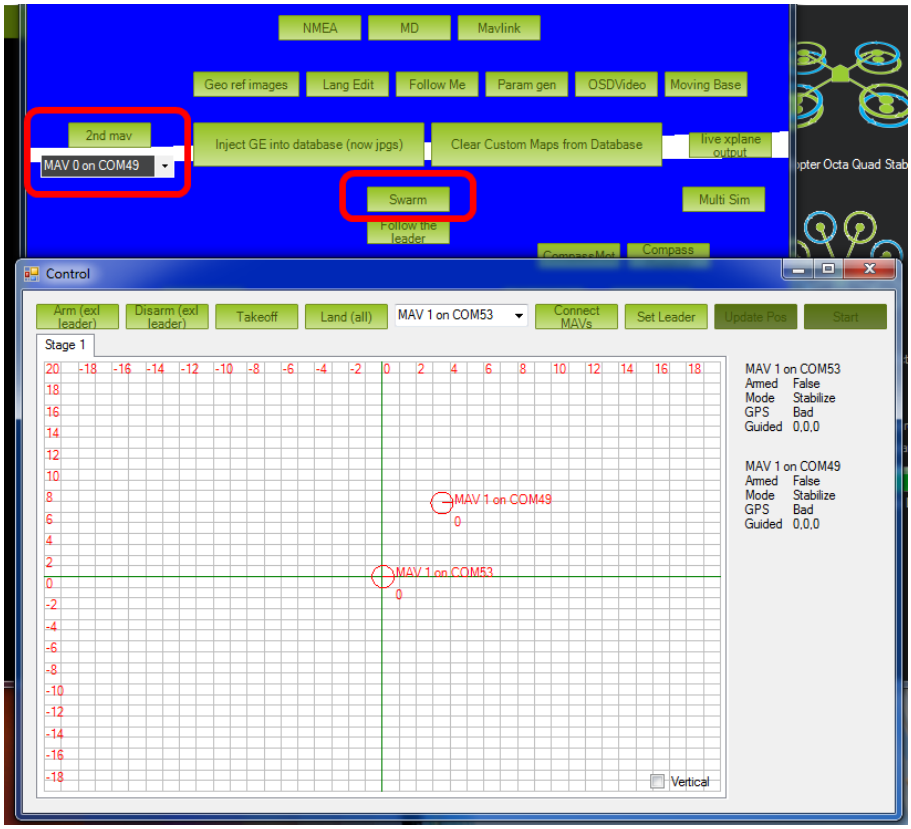

Figure 2. Mission Planner Swarm Application With Grid Offset [3]

Though Mission Planner cannot connect to multiple vehicles simultaneously without the swarm application, there are other GCSs that can. APM Planner 2.0 and Q Ground Control are two GCSs investigated that support multiple vehicle connections. Further investigation found that APM Planner 2.0 was relatively new and did not support Python, a programming language that Mission Planner did support. Q Ground Control was found to not have a user friendly interface, and not much support was found on forums.

Flocking

Flocking has been simulated in several different scenarios. The Boid algorithm is commonly used as a flocking algorithm [12]. The most basic Boid scheme involves three rules for application: separation, alignment, and cohesion. The flock must separate to avoid colliding with flock occupants, align towards the average heading of flock occupants, and form a cohesive

flock by moving towards the geometric center of the flock. This flocking algorithm has been modified and simulated using a variety of algorithms and programming environments. Implementing this behavior into SUAS would greatly enhance autonomy with the use of cooperative behavior and control by eliminating the need for manual control. The algorithms force the operation of the vehicles in the flock to depend on every other vehicle in the flock. By appropriately weighting the various rules, to include the possible addition of more rules associated with target seeking or obstacle avoidance, the vehicles can essentially create their own steering commands and sustain navigation. Figure 3 below shows a visual of flocking in Python [13].

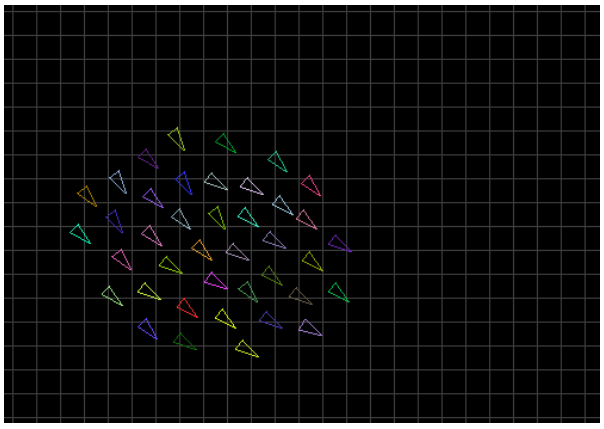

Figure 3. Flocking Algorithm Run Through Python [13]

Flocking can be a beneficial component of mission effectiveness. In recent research, a methodology was modified from Craig Reynold's model in order to provide the most ideal flock flight formation for fuel saving and mission endurance [14]. Reynold's model focus of collision avoidance, flock centering, and velocity matching was restructured to focus on precise positioning in relation to the Formation Geometry. Precise positioning allowed for the flock members to maintain their offset during navigation. In close proximity, cooperative behavior and control as well as timely sensing and communication are essential between the aircraft in the flock to prevent collision or mission failure. The aircraft must interact by exchanging

information among the other aircraft in the flock such as velocity and position. This information sharing can be accomplished through such measures as a local area network, which will prevent the flock from needing central guidance, initiating semi-autonomous behavior. The flock formation was designed as a staggered "V" pattern in order to minimize drag.

Collision avoidance was implemented by establishing a buffer between the aircraft in the flock. If the distance between two aircraft in the flock was within 75% of the established offset, the aircraft would speed up or slow down accordingly to avoid collision and maintain the offset balance [14].

If the leader in the flock maintained its position in the flock for an extended amount of time, the leader aircraft autonomously switched positions with another aircraft in the flock [14]. This autonomous leader switch benefited the mission endurance by balancing drag savings and fuel usage over time across the vehicles in the flock.

Through these methodological principles, the optimum drag reduced flock configuration still provided too much of a collision risk. However, longitudinal spacing provided less drag effect than lateral spacing. Therefore, the flock configuration implemented further longitudinal spacing in order to maintain reduced drag effects, while also decreasing the collision risk. Through this configuration, a 9.7% reduction in drag was achieved allowing for a 14.5% increase in mission endurance [14].

Rover/Relay

For overcoming communication range limitations, cooperative behavior has been demonstrated through use of rover and relay vehicle configurations. Using a rover and relay SUAS, with incorporated cooperative behavior and control algorithms, anticipated range was able to increase over 50 percent [15]. With moving target search areas, the algorithm was able increase coverage area by over 110 percent. The relay aircraft acted as a messenger aircraft,

sending valuable information from the GCS to the mission oriented rover aircraft, as well as transmit rover information to the GCS.

Equation 1 is used to calculate the anticipated position of the rover aircraft in latitude and longitude coordinates, given the actual rover position, P_{actual}, in latitude and longitude [15]. The speed lead factor, C_{SL}, weights the amount of system speed lead. The heading, *h*, and ground speed, *v*, in meters per second also contribute to the anticipated rover position. The constant, *mpD*, measures the meters per degree latitude and longitude for the location.

$$P_{anticipated} = P_{actual} + \left(\frac{C_{SL}}{10}\right)\begin{bmatrix}\sin h \\ \cos h\end{bmatrix}(mpD)v\Delta T \quad \textbf{(1)}$$

Then, the midpoint of the rover aircraft is calculated using Equation 2. The position of the GCS, P_{GS}, is used in the algorithm [15].

$$P_{mid} = \frac{P_{anticipated} + P_{GS}}{2} \quad \textbf{(2)}$$

With the midpoint of the rover aircraft calculated, the distance to the midpoint from the GCS could be calculated from Equation 3 [15].

$$D = \left|2\left(\frac{P_{mid} - P_{GS}}{2}\right) * mpD\right| \quad \textbf{(3)}$$

Finally, the position of the relay is able to be calculated in Equation 4, given the weighted average formula in Equation 5 [15].

$$P_{relay} = \frac{\sum_{i=0}^{\#activeRovers} W_i * P_{mid}}{\sum_{i=0}^{\#activeRovers} W_i} \quad \textbf{(4)}$$

The weighted average in Equation 5 requires the unsigned positive integer Distance Bias Factor, C_{db} [15].

$$W = D^{\frac{C_{db}}{10}} \quad (5)$$

It was later found from testing during the research, that the calculated midpoint of the rover was the optimum location for the relay aircraft to loiter for maximum range [15]. It was also found that smaller loiter radii coupled with slower and more maneuverable relay aircraft increased the overall communication range between the GCS and the rover aircraft.

Surveillance

Cooperative control has also been used for surveillance. Past research discusses cooperative control algorithms used for multiple SUAS to perform surveillance using equal angular spacing from the ground target [16]. The algorithm allows each SUAS in the surveillance mission to loiter around a target at an equal angular spacing from one another on the same loiter path. This loiter would allow for the target to continually be within the Field of View (FOV) of at least one SUAS camera. It was found that roll had the largest impact on the FOV of the fixed body camera. Also, wind greatly affected the visibility time of the camera. The wind's effect was measured with the wind at speeds of 0-50% of the vehicle air speed. It was found that at wind speeds greater than 50% of the nominal airspeed, the visibility time of the camera was too short to be considered mission effective. It was also found that the more vehicles operating in the mission, the less wind affected the visibility time. This low wind effect is because there were more operational cameras focusing on the target during the mission. This use of multiple vehicles to limit wind effects highlights why cooperative control can be important to the mission. The multiple SUAS need to obtain current position, velocity, angle from target, and other flight information from the other SUAS in the mission to affect its own course. In this regard, the SUAS need to work together for the most effective solution.

Metrics

Measuring autonomy is perhaps another challenge, since it is mostly known as a subjective evaluation. This research focuses on implementing cooperative behavior and control. However, cooperative behavior and control is a small part of autonomy. Cooperative behavior and control can improve autonomy. Nevertheless, autonomy must somehow be able to be measured in order to evaluate SUAS capabilities. Bruce Clough, from the Air Force Research Laboratory (AFRL), introduced an Autonomous Control Level (ACL) chart in order to measure autonomy [17]. Clough points out that automatic and autonomous are not the same. *Automatic* means that the system will follow directions exactly as specified. *Autonomy* means the system has free will or choice outside of influence. Clough integrated existing autonomous evaluation categories from other autonomy scales in order to create his own ACL. The ACL categorizes SUAS on a scale of zero to ten, ten being fully autonomous like a human, and zero being a Remotely Piloted Aircraft (RPA). Integrating human dynamists' Observe, Orient, Decide, and Act (OODA) measures, columns were made measuring perception/situational awareness, analysis/coordination, decision making, and capability of SUAS. Table 1 shows Clough's ACL, which could be used for data measurement and analysis between the cooperative behavior and control methods presented in this research.

Table 1. Clough's Autonomous Control Level (ACL) Chart [17]

Level	Level Descriptor	Observe Perception/Situational Awareness	Orient Analysis/Coordination	Decide Decision Making	Act Capability
10	Fully Autonomous	Cognizant of all within Battlespace	Coordinates as necessary	Capable of total independance	Requires little guidance to do job
9	Battlespace Swarm Cognizance	Battlespace inference – Intent of self and others (allies and foes). Complex/Intense environment – on-board tracking	Strategic group goals assigned. Enemy strategy inferred	Distributed tactical group planning. Individual determination of tactical goal. Individual task planning/execution. Choose tactical targets	Group accomplishment of strategic goal with no supervisory assistance
8	Battlespace Cognizance	Proximity inference – Intent of self and others (allies and foes) Reduced dependence upon off-board data	Strategic group goals assigned. Enemy tactics inferred. ATR	Coordinated tactical group planning. Individual task planning/execution. Choose targets of opportunity	Group accomplishment of strategic goal with minimal supervisory assistance (example: go SCUD hunting)
7	Battlespace Knowledge	Short track awareness – History and predictive battlespace data in limited range, timeframe, and numbers. Limited inference supplemented by off-board data	Tactical group goals assigned. Enemy trajectory estimated	Individual task planning/execution to meet goals	Group accomplishment of tactical goal with minimal supervisory assistance
6	Real Time Multi-Vehicle Cooperation	Ranged awareness – on-board sensing for long range, supplemented by off-board data	Tactical group goals assigned. Enemy location sensed/estimated	Coordinated trajectory planning and execution to meet goals – group optimization	Group accomplishment of tactical goal with minimal supervisory assistance. Possible close air space separation (1-100 yds)
5	Real Time Multi-Vehicle Coordination	Sensed awareness – Local sensors to detect others, Fused with off-board data	Tactical group plan assigned. RT Health Diagnosis; Ability to compensate for most failures and flights conditions; Ability to predict onset of failures (e.g. Prognostic Health Mgmt). Group diagnosis and resource management	On-board trajectory replanning – optimizes for current and predictive conditions. Collision avoidance	Group accomplishment of tactical plans as externally assigned. Air collision avoidance. Possible close air space separation (1-100 yds) for AAR, formation in non-threat conditions
4	Fault/Event Adaptive Vehicle	Deliberate awareness – allies communicate data	Tactical plan assigned. Assigned Rules of Engagement. RT Health Diagnosis; Ability to compensate for most failures and flight conditions – inner loop changes reflected in outer loop performance	On-board trajectory replanning – event driven. Self resource management. Deconfliction	Self accomplishment of tactical plan as externally assigned. Medium vehicle airspace separation (100's of yds)
3	Robust Response to Real Time Faults/Events	Health/status history & models	Tactical plan assigned. RT Health Diag (What is the extent of the problems?). Ability to compensate for most control failures and flight conditions (i.e. adaptive inner-loop control)	Evaluate status vs required mission capabilities. Abort/RTB if insufficient	Self accomplishment of tactical plan as externally assigned
2	Changeable Mission	Health/status sensors	RT Health diagnosis (Do I have problems?). Off-board replan (as required)	Execute preprogrammed or uploaded plans in response to mission and health conditions	Self accomplishment of tactical plan as externally assigned
1	Execute Preplanned Mission	Preloaded mission data. Flight Control and Navigation Sensing	Pre/Post Flight BIT. Report status	Preprogrammed mission and abort plans	Wide airspace separation requirement (miles)
0	Remotely Piloted Vehicle	Flight Control (attitude, rates) sensing. Nose camera	Telemetered data. Remote pilot commands	N/A	Control by remote pilot

Another metric used in previous research for Unmanned Vehicles (UV) involved including human, UV, and the interaction measures. The five groups of metrics are shown in Table 2 [18].

Table 2. Unmanned Vehicle Human Supervisory Control Metric Classes and Subclasses [18]

Mission Effectiveness	(e.g., key mission performance parameters)
UV Behavior Efficiency	(e.g., usability, adequacy, autonomy, reliability)
Human Behavior Efficiency	-Attention allocation efficiency (e.g., task switching times, prioritization) -Information processing efficiency (e.g., decision making accuracy, reaction times)
Human Behavior Precursors	-Cognitive Precursors (e.g., SA, mental workload, self-confidence, emotional state) -Physiological Precursors (e.g., physical comfort, fatigue)
Collaborative Metrics	-Human/UV Collaboration (e.g., trust, mental models) -Human/Human Collaboration (e.g., coordination metrics, team mental model, team SA) -UV/UV Collaboration (e.g., vehicle reaction times to situational events that require autonomous collaboration

UV behavior involves usability, adequacy, autonomy, and reliability. Usability is associated with efficiency, memorability, errors, and user satisfaction [19]. Adequacy is characterized by the impact on mission support and is composed of autonomy, accuracy, and reliability.

Human behavior involves the mission choices and actions made to satisfy the objective. Human behavior is categorized into attention allocation efficiency and information processing efficiency [19]. Attention allocation is measured through efficiency across tasks and involves task switching times and prioritization and could be affected with increased workloads.

Information processing results from the ability to dissect and understand the tasks of the mission and involves the decision making accuracy, and reaction times.

Human Behavior Precursors consist of processes that occur before a recognized action or result [19]. These include cognitive precursors and physiological precursors. Cognitive precursors involve the social or psychological factors, while the physiological precursors involve physical factors, such as fatigue and physical discomfort.

Collaborative Metrics are measured through the interaction between operators and UVs [19]. With multiple vehicles controlled by a single operator, the collaboration between the UV's is also considered. The vehicles must interact with one another for cooperative behavior and control and must pass and receive information. Therefore, the reaction times required for these messages could be measured for efficiency. The interaction between human and the vehicles must also be measured through the trust that the human has in the vehicle and mental models. If too much trust is given in the vehicle's operation, the risk of complacency appears. However, if too little trust is put into the vehicles, the potential capability of the vehicle is never fully realized. With multiple operators, human to human collaboration is also measured. Cooperation and team building exercises are vital to working as a team. Therefore, mental human behavior analysis models could help measure the social skills involved with the team members.

Mission Effectiveness measures how well the overall system meets its objectives [19]. These involve key mission performance parameters. If mission effectiveness is high and lower level measures of performance are low, either the measures of performance are not appropriate or the measure of effectiveness is not measuring what is important.

With a single operator and multiple unmanned vehicles, an architecture was created of the metrics and where they are measured within the system. [18]. The architecture starts with Human Behavior Precursors, which affect Human Behavior Efficiencies, which affect UV Behavior Efficiencies. Then, UV Behavior Efficiencies are cycled back to Human Behavior

Efficiencies with usability, adequacy, autonomy, and reliability information. The information processing efficiency is then measured.

Past research shows a way of measuring the effectiveness, efficiency, and complexity of flocking UAVs [20]. Effectiveness is measured by the amount of targets killed, K, out of the number of enemy events, E, using Equation 6. Efficiency is measured by the amount of targets killed out of the number of munitions launched, LM, in Equation 7.

$$effectiveness = \frac{K}{E} \tag{6}$$

$$efficiency = \frac{K}{LM} \tag{7}$$

The complexity can also be measured by the mean length of the UAV's target list. The longer the target list, the more complicated the algorithm implementation and on-board processing becomes. A long target list introduces more variables to analyze and requires higher levels of on-board processing to sort through large data sizes. Dudek's Taxonomy [20] was also used, displayed in Table 3, to measure different flocking UAV attributes.

Table 3. Dudek's Taxonomy Properties Used In The Experiments [20]

Axis	Subdivision	Value/Remarks
Collective size	SIZE-ALONE	1
	SIZE-PAIR	2
	SIZE-LIM	3-10
	SIZE-INF	N/A
Communication Range	COM-NONE	0
	COM-NEAR	10,000m
	COM-INF	1e10m
Communication Topology	TOP-BROAD	Used always
	TOP-ADD	N/A
	TOP-GRAPH	N/A
	TOP-TREE	N/A
Communication Bandwidth	BAND-ZERO	Same as COMM-NONE
	BAND-LOW	Not used
	BAND-MOTION	Self-created target list
	BAND-INF	Entire target list
Collective Reconfigurability	ARR-STATIC	Dependent on UAV velocity which is
	ARR-COM	dependent on number of group size
	ARR-DYN	
Processing Ability	PROC-SUM	N/A
	PROC-FSA	N/A
	PROC-PDA	N/A
	PROC-TME	Used always
Collective Composition	CMP-IDENT	Used
	CMP-HOM	Same as CMP-IDENT
	CMP-HET	used

The values in the table are sample values used in the research, but the categories present opportunities and areas to evaluate for cooperative behavior and control.

Summary

This chapter has introduced several related research efforts, both past and present. The need for cooperative behavior and control can be seen through government Request For Proposals (RFP) and through government organizations, such as DARPA, and their pursuit of swarm technology programs.

Military policy prohibits military flight outside of reserved training sites, as well as multiple vehicle operation from a single GCS. The lack of aerial vehicle operation from a single GCS, forces other means of communication, such as through a network. However, latency issues may now be introduced. With the use of ground vehicles and a multi-rotor, heterogeneous vehicle cooperative behavior and control can be demonstrated, which will hopefully aid in military flight policy adjustments. Policy restricting one aerial vehicle per GCS presents an opportunity for this research to adjust current restrictions. The policy addresses an investigative question in reference to the challenges of using heterogeneous vehicles from a single GCS. Testing cooperative behavior and control with multiple rover vehicles on a single GCS gives data that may support the claim of using multiple aerial vehicles on a common GCS. This policy also leads to investigations or experimentations in this research as to whether operating multiple vehicles from a single GCS is really beneficial over using separate GCSs.

Though cooperative behavior and cooperative control may seem similar, they are different concepts. Cooperative behavior involves the collective acts performed by a group of individuals, while cooperative control demonstrates how the system is run or managed. Cooperative control, the "how", often determines the cooperative behavior, the "what." Defining these terms helps communicate this research and its intent.

Past research in cooperative control has offered future improvements with the addition of swarm applications in open source GCS software, such as Mission Planner. The application offers simultaneous vehicle connection to a common ground station, previously incapable with

Mission Planner. This simultaneous connection to a single ground station leads the way for further modifications, such as leader-follower offset re-configuration, and the inclusion of new behavior modifications. Though other GCS software exists that support multi-vehicle connections, Mission Planner is used through the extent of this research due to its Swarm application, Python capability, and appears to be the most stable and supported GCS in the community through the documentation offered by users. However, one of the leading limitations of Mission Planner is that it cannot connect to multiple vehicles, aside from what was seen in the Swarm application.

The autopilot used in this research is the Pixhawk, which offers higher processing capabilities than other related low cost autopilots. In terms of this research, most of the processing will be done from the GCS, but having a Pixhawk will hopefully reduce latency.

Surveillance is another type of cooperative control that could benefit with the use of multiple vehicles. With this research, vehicle orientation and communication is paramount. The methodology used to set equal distances between vehicles could possibly be used with heterogeneous vehicles towards a common target or goal. Therefore, this research could aid in the implementation of vehicle offset of multiple vehicles. Surveillance is a possible application that can be investigated further through camera calculations to find the optimum settings for a multi-rotor camera following a rover.

Previous research illustrates that to measure the performance of unmanned vehicles, more than the vehicles' mission capability needs to be analyzed. Human performance and supervisory reaction needs to be incorporated to measure the system as a whole. With multiple vehicles, and a single operator the vehicle collaboration will need to be measured through reaction times or latency as well as the human to vehicle interaction through degrees of trust from the operator towards the vehicle.

III. Methodology

Chapter Overview

Using a collection of past, current, and new developmental research, a set of test procedures can be established and implemented. This chapter discusses the materials and equipment to be used in the research test procedures, the test procedures involved with the research, and the data measures to be collected. The aim of this chapter is to communicate the architectures, hardware, and constructive test procedures that adequately address the following investigative questions out of the six total investigative questions associated with the research.

2. What are the challenges of using multiple heterogeneous vehicles from a single GCS?
3. What is the initial architecture that can be implemented and improved upon?
4. What appropriate assessment measures should be used for analysis?
5. What are the performance limitations given current architecture?
6. What cooperative behavior applications are reasonable or achievable given current limitations?

With proper test procedures set in place, responses to the investigative questions should ultimately lead to conclusive solutions to the research objective.

Materials and Equipment

As part of this research, the system is comprised of heterogeneous unmanned vehicles, including ground and air vehicles, and a GCS. Each vehicle in the system is its own system, classifying the system as a system of systems. A system of systems occurs when each system within the system can operate independently without the use of the other systems. However, each system in the system can also operate together to reach a common goal, hence the name system of systems. Therefore, each vehicle must have components, consisting of an autopilot,

communications, propulsion, and battery, to operate it and to exhibit cooperative behavior and control with the other vehicles in the system.

Autopilot

The autopilot is the "brain" of the vehicle. It interprets commands and distributes them to the rest of the components. Without an autopilot, the vehicle cannot function autonomously. For the Pixhawk autopilot, these autonomous functions include waypoint navigation, loiter points, Guided mode, and failsafe implementations [21]. A set of waypoints or loiter points edited from GCS software can be loaded onto the autopilot. The loiter points are used for aerial vehicles. Guided mode makes the vehicle move towards a set point. A Fly-to-Here function, a variant of Guided mode, allows the user to mark a point on the GCS software's map, while connected to the vehicle, for the vehicle to immediately navigate towards. Once the vehicle arrives at the point, the vehicle will loiter about that point. When the vehicle's battery is low or telemetry reception is lost, failsafes on the autopilot allow for the vehicle to autonomously land or return to a home location. Most of the vehicle components are connected to the autopilot, some even directly powered by the autopilot's output voltage. The autopilots are not unique to a specific vehicle type, meaning they can be used by both ground and air vehicles by loading the desired vehicle's firmware onto it. Past research has used the ArduPilotMega (APM) version 2.5 autopilot, due to its low cost, accessibility, similarity to fielded systems, and flexible use as an open source platform [22]. Firmware loaded onto the APM and Pixhawk is available in the open community. This research is using the newer 3DR Pixhawk autopilots for the vehicles, as shown in Figure 4 [21]. The Pixhawk carries most of the same attributes as the APM, but has a faster processor, potentially allowing for the incorporation of new cooperative behavior and control techniques.

Figure 4. 3DR Pixhawk Autopilot [21]

Telemetry Modems

In order for the autopilots and the vehicles to communicate with the GCSs, telemetry modems are necessary. One telemetry modem is connected to the autopilot, while the coupled modem is connected to the GCS. Telemetry, such as vehicle parameters, is transferred through each pair of modems. These modems create the wireless connection or link between the vehicle and GCS. The telemetry modems used in this research are 915 MHz 3DR radios, shown in Figure 5 [23]. The Net ID of these modems can be changed so each modem will only communicate with a modem with the same Net ID. When multiple pairs of modems are used simultaneously, the Net IDs of each pair must be set different from one another. If not, one modem may connect to a modem from a different pair of modems. For lower latency performance, the Max Window can be changed to lower values, down to 33 ms. The Max Window setting controls how often telemetry packages and control commands are sent back and forth

between a pair of modems. With a 33 ms setting, one telemetry package would be sent from the autopilot modem to the GCS modem every 33 ms.

Figure 5. 915 MHz 3DRobotics Telemetry Modems [23]

Ground Vehicles

Rover ground vehicles are used in this research. They are modified hobbyist vehicles owned by AFIT. The base and structure of the vehicles are Traxxas vehicles [24]. The autopilot, telemetry modems, voltage regulator, GPS, and receiver were added to give the vehicle autonomous capabilities. Further description of these modifications can be seen in Appendix A. Seen below in Figure 6, they consist of the same autopilot and communication components as the air vehicles in the system. The wheels have shocks to absorb the force of the vehicle weight and terrain variations. The vehicles can reach speeds of up to 60mph, but high speeds are greatly suppressed through the use of speed limit settings in the autopilot software, to prevent overturning the vehicles.

Figure 6. Rover Ground Vehicles

Air Vehicles

The air vehicles used in the research are multi-rotors. These vehicles, seen in Figure 7, are AFIT owned and bought off-the-shelf from 3DRobotics [25]. The X8s include a Pixhawk autopilot, a 3DRobotics GPS/Compass, a speed controller, motor, and a pair of 3DRobotics telemetry modems. A more detailed look into the components of the vehicle can be seen in Appendix B. These aerial vehicles are more suited for this research than planes because multi-rotors are more maneuverable in small areas than planes and aren't necessarily subjected to the same constraints as planes. Planes offer higher safety risks, which must maintain a certain elevation for safety, than multi-rotors, which could operate at eye level if need be. Demonstrating cooperative behavior and control implementation on multi-rotors may lay foundations for plane integration.

Figure 7. X8 Multi-Rotor [25]

Ground Control Station

The user interacts with the vehicles through a GCS. GCS software acts as a mission planning element and Heads Up Display (HUD) for the user. There are many types of GCS software, but Mission Planner [26] and Droid Planner 2 [27] are used in this research. Mission Planner is an open source GCS software readily available to the public. Its open source characteristic offers a low cost operational capability, with an active support forum due to its popularity and use. Mission Planner allows the operator to create and edit waypoints or loiter points for a vehicle as well as set vehicle parameters. Mission Planner's Fly-to-Here function, discussed previously, puts the vehicle into Guided mode and forces the vehicle to follow a given point. Vehicle gain settings can be tuned through Mission Planner as well. Mission Planner saves telemetry logs (T-logs) with the connection of a vehicle. These T-logs can be managed through Mission Planner and record vehicle navigation and other established vehicle parameters. Other open source GCS software is available, some even with the ability to simultaneously connect to multiple vehicles. However, other features are missing or are still in development that Mission Planner has included. For instance, Mission Planner has the ability to run Python scripts. Python is a programming language like Java. With the ability to run Python scripts, the user can add or incorporate new behaviors or controls into Mission Planner and the vehicle(s) it is

connected to. This Python capability adds a customizable aspect to the GCS software and allows the user to shape the operation to a specific application. Mission Planner uses Google Maps and acts as a HUD, displaying all necessary and optional vehicle parameters to the user, such as vehicle GPS coordinates, local time, battery power, waypoints, altitude, heading, and other flight instruments, as shown in Figure 8 [26].

Figure 8. Mission Planner

Droid Planner 2 is an open source free smart phone or tablet application available through Google's Play Store. Documentation on the application is supported from 3DRobotics [28], the same company that sells the autopilots and telemetry modems used in this research. The application requires GPS access on the smart phone or tablet that it's installed on, internet access for the Google maps, and a pair of telemetry modems. Both telemetry modems must have the same Net ID in order for the modems to communicate between one another. One modem is plugged into the smart phone or tablet and the other to the autopilot of the vehicle. The version of the application that will be used with this research is Droidplanner_v2.8.6_RC3. Unfortunately,

this application does not support ground rover vehicles as of yet. The application only supports multi-rotor and possibly plane aerial vehicles. However, multi-rotor aerial vehicles are the only vehicles seen tested with the application.

Once both modems are physically connected to the phone/tablet and autopilot and the autopilot/vehicle turned on, the application can make a link connection through the "Connect" button seen on the application map. At that point, seen in Figure 9, five buttons appear on the bottom of the application labeled "Edit, Home, Land, Loiter, Follow." The "Edit" button is used for editing waypoints for the vehicle. The "Home" button is for returning the vehicle to home location. The "Land" button is for landing the vehicle. The "Loiter" button is for the vehicle to loiter around a location. The "Follow" button sends the vehicle into a Follow-Me Mode and allows the vehicle to follow the GCS where the application is operating, in this case the smart phone or tabled the application is installed on [27]. With the multi-rotor, the altitude the multi-rotor is at when transitioned to Follow-Me Mode will be the altitude the multi-rotor maintains in Follow-Me Mode. However, the vehicle must be in Guided Mode before the "Follow" command can execute properly. The modes of the vehicle can be changed from the displayed current mode in the top right of the application. Also, the different menu tabs, similar to the menu tabs at the top of Mission Planner, can be accessed in the top left of the application from a drop down menu. In the Editor menu, waypoints can be written for the vehicle. In the Parameters menu, vehicle parameters can be edited. The application will save Telemetry Logs (T-logs) to a specified folder on the device after a mission, just like with Mission Planner. When in Follow Me Mode, the vehicle will follow the GCS wherever it is moved. The blue dot that appears on the application's map is the device's location and the orange arrow is the vehicle's location. Whenever in Guided Mode, the destination appears as a green dot. Therefore, in Follow-Me Mode, the green dot should appear in place of the blue dot, the device's location. For the purposes of this research, Follow Me Mode will be the primary utilization for Droid Planner 2.

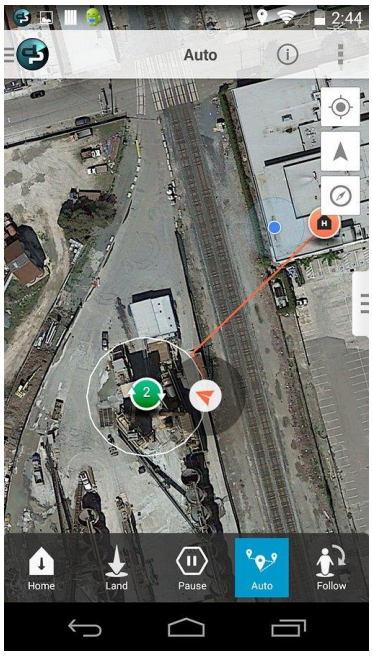

Figure 9. Droid Planner 2 [27]

Configuration Architectures

Architectures were defined for the different heterogeneous vehicle configurations used in this research. These architectures include SV-1s and a SV-4 according to the Department of Defense Architecture Framework (DoDAF) [29]. An SV-1 describes the systems interface. "Systems, system items, and their interconnections" are represented in the SV-1 [29]. The SV-4 describes systems functionality. "The functions (activities) performed by systems and the system data flows among system functions (activities)" are seen in the SV-4 [29]. The software used to build the architectures was Enterprise Architect [30].

There are four different vehicle-GCS configurations used throughout this research. Displayed in sequential order of procedure, the first configuration involves using Mission Planner's Swarm application on a single GCS between two vehicles. This SV-1 of the configuration is seen in Figure 10. A GPS receiver is connected to the autopilot in each vehicle. The GPS signal is sent from the GPS satellites to the GPS receiver and then to the vehicle's Pixhawk autopilot. The vehicles are connected to the GCS through a 915 MHz telemetry link

from the telemetry modems. The Mission Planner Swarm application is used as the cooperative behavior and control method, which is a part of Mission Planner. The operator will manually control the leader vehicle with a radio during testing. Once the swarm application is started, the application overrides operator control of the vehicle. This override prevents the need for a follower vehicle radio, but still requires a leader vehicle radio for control. The operator will control the offset and cooperative behavior and control method through the GCS. A Google map server uses internet via Wi-Fi to load maps onto Mission Planner.

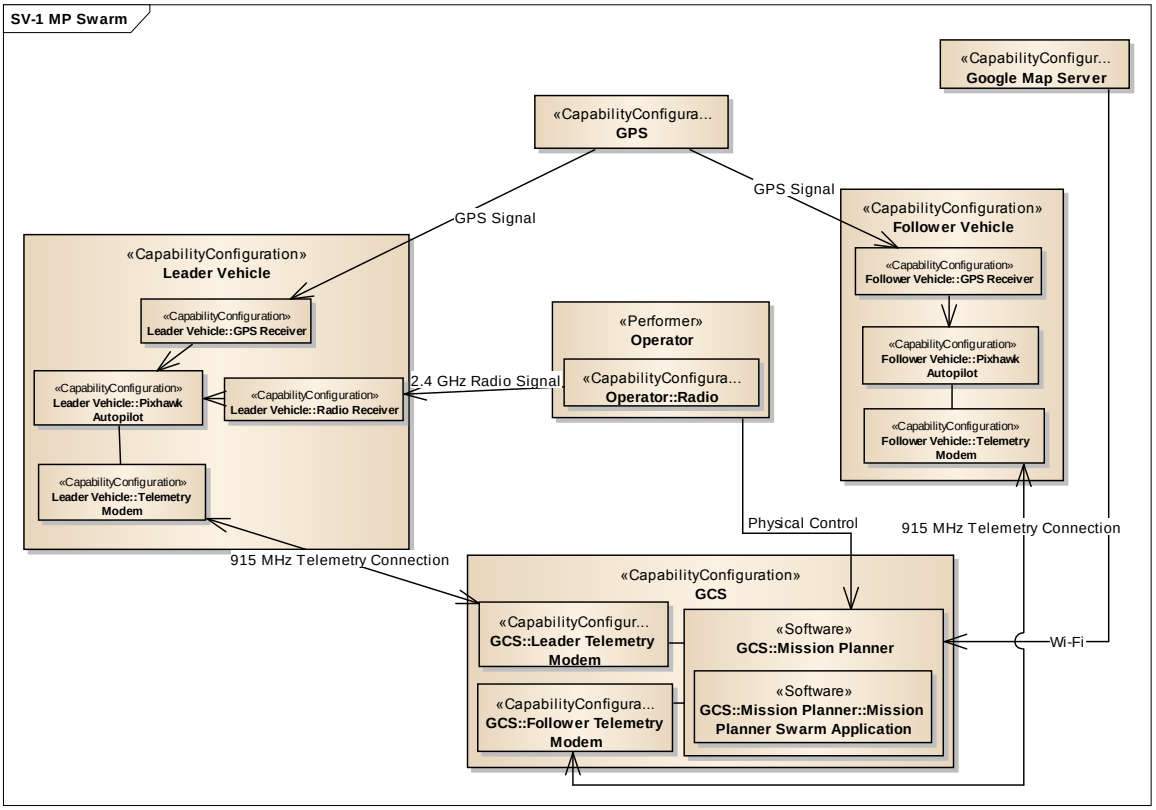

Figure 10. Mission Planner Swarm SV-1 (Configuration 1)

The SV-1 in Figure 11 shows the second configuration, consisting of the Python script used to mimic the performance of Mission Planner's Swarm application. This configuration will have both vehicles being operated from a single GCS. Connections are similar to Figure 10, except a Python script for each vehicle is run simultaneously across two instances of Mission Planner on the same GCS. There is a radio for the vehicle of each operator. The follower vehicle

requires a radio for safety reasons or so the safety pilot can switch vehicle modes and kill the Python script if necessary.

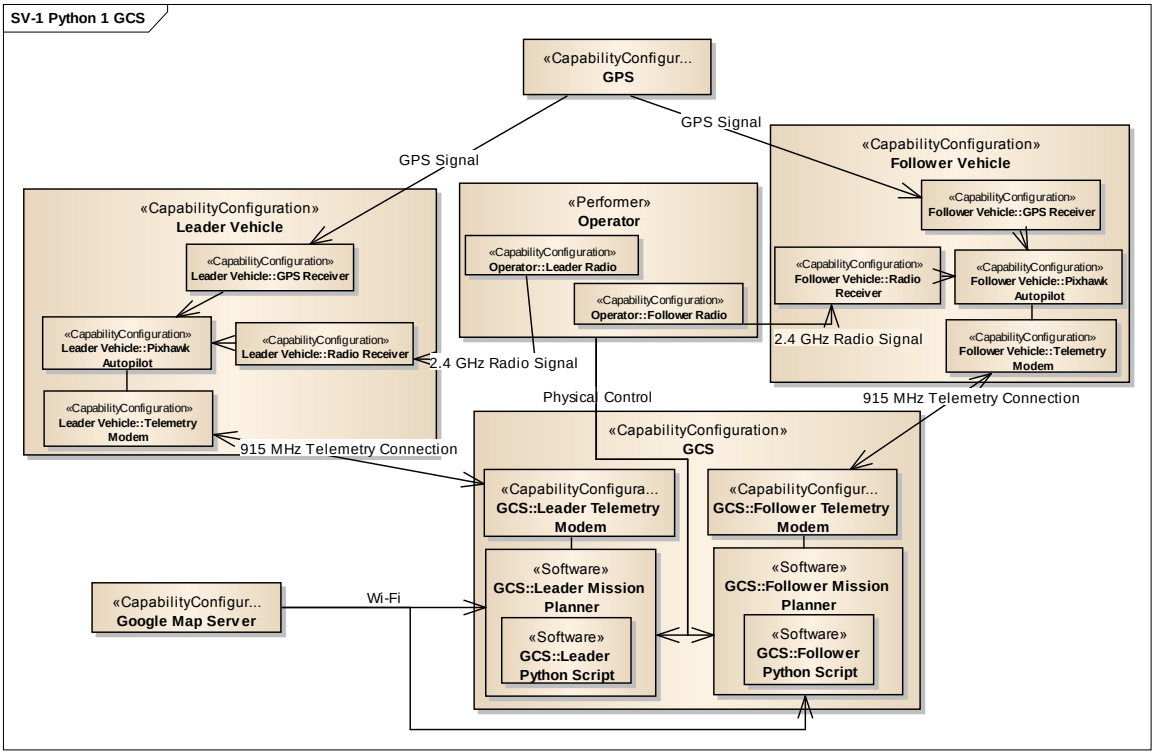

Figure 11. Python Method on One GCS SV-1 (Configuration 2)

The SV-1 of the Python method running on separate GCSs is seen in Figure 12. The only difference from Figure 11 is that now each vehicle script is run from a single instance of Mission Planner on separate GCSs. The GCSs are linked through a Wi-Fi connection introduced in the Python scripts.

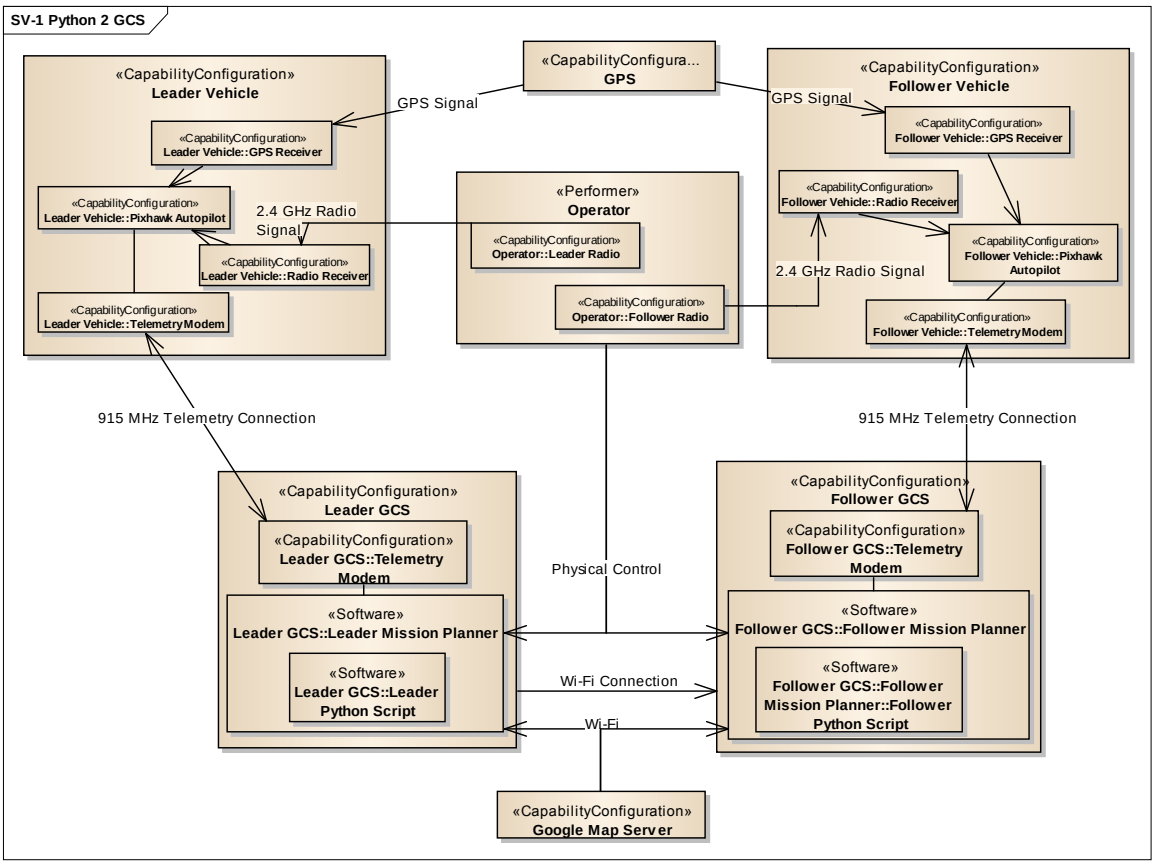

Figure 12. Python Method on Separate GCSs SV-1 (Configuration 3)

The fourth and final SV-1 in Figure 13 involves the Droid Planner 2 application as the follower vehicle's GCS software. This application is operated from a smart phone. The leader vehicle is still operated from Mission Planner, but only for changing vehicle parameters and creating waypoints for accuracy tests. Manual control of the leader vehicle will only be needed for latency tests. Accuracy tests will require the leader vehicle to operate autonomously through loaded waypoints. The follower GCS, or phone, will be attached to the leader vehicle to execute a "Follow Me" capability. This capability will allow the follower vehicle to follow the leader vehicle, due to the GCS's attachment to the leader vehicle. The smart phone must have access to the internet for maps and GPS enabled due to Droid Planner 2 requiring a GPS signal. This signal is

separate from the vehicle's GPS signal and shows where the GCS is on the map of the Droid Planner 2 application.

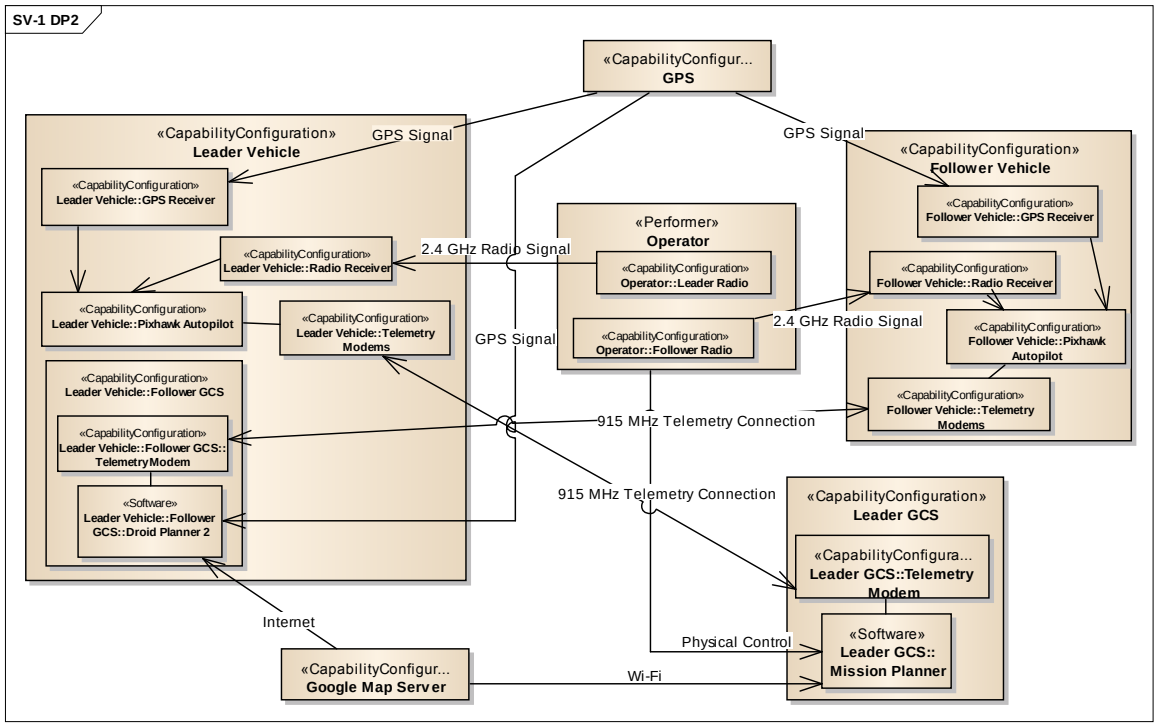

Figure 13. Droid Planner 2 Method SV-1 (Configuration 4)

The SV-4 shows the hierarchical functions of the system nodes involved with all SV-1s. Figure 14 shows the SV-4 of the vehicles involved in the experiment configurations. The navigation functions involve determining the vehicle's position using GPS, updating waypoint(s) or vehicle gain/settings, and steer or loiter at a waypoint. The communicate function involves receiving commands from the operator's radio, receiving commands from telemetry modems, sending telemetry information, receiving GPS signals, and recording log files. The modems' connection between the autopilot and GCS allows for telemetry sharing. Some vehicles will acquire an optional camera or video camera for imagery.

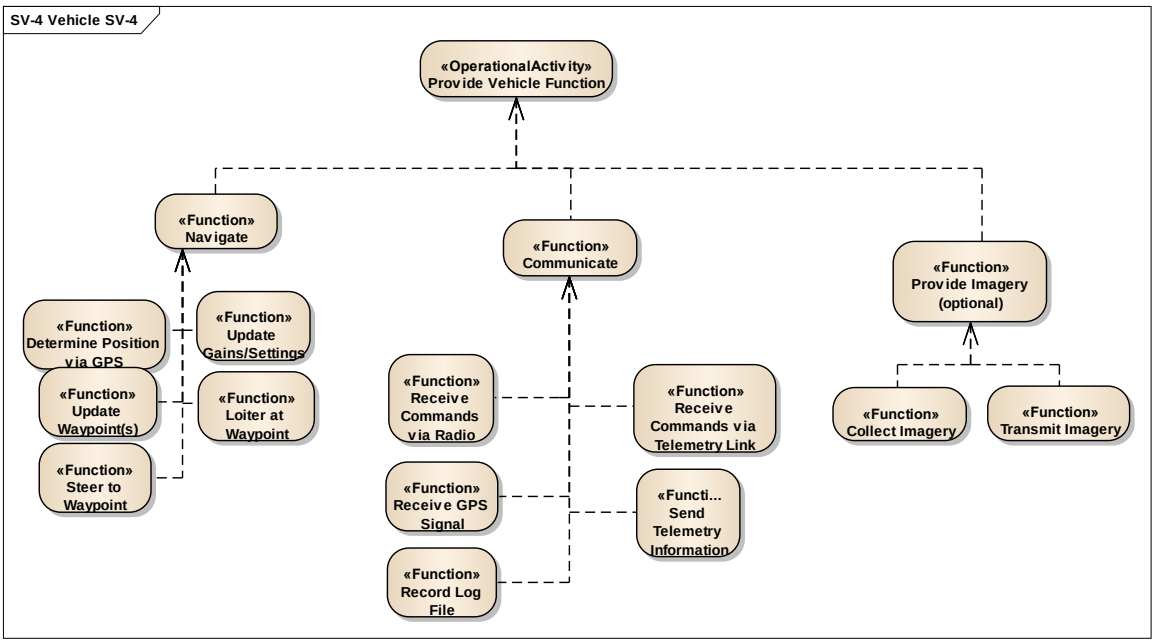

Figure 14. Vehicle System Node Functions SV-4

The functions of Mission Planner are seen from the SV-4 in Figure 15. The swarm application can be accessed through Mission Planner's Ctrl + F command. This application allows for follower vehicles to be connected to the same instance of Mission Planner as a leader vehicle. An offset can be placed between the leader and follower vehicles by placing their location on an offset grid.

Waypoints can be edited or written through Mission Planner. They can also be loaded onto Mission Planner from a vehicle. Google maps are used so that Mission Planner can display the vehicle's location.

Parameters can be set on Mission Planner that involve waypoint radius, vehicle cruise speed, modem telemetry rate, and modem max window. These parameters are the ones used for this research.

Mission Planner records telemetry logs (T-log) throughout the connection to a vehicle. These T-logs record vehicle navigation and parameters throughout the vehicle's connection. Not

only are these T-logs recorded by Mission Planner, but they can be loaded and played back through Mission Planner or converted to Excel files to organize parameter settings or data.

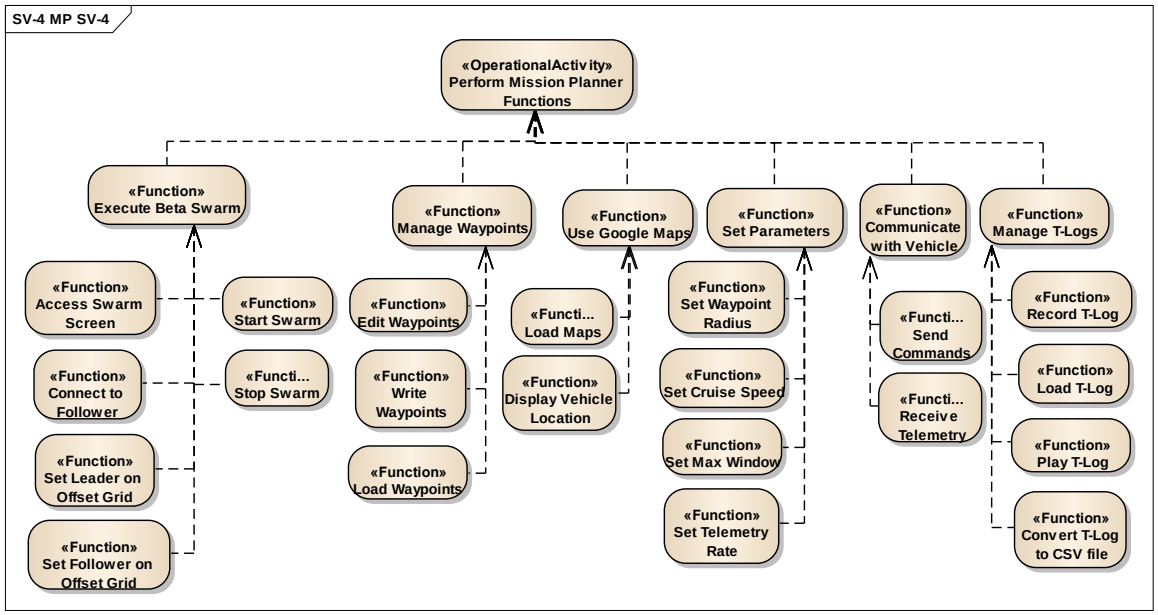

Figure 15. Mission Planner System Node Functions SV-4

The functions of the leader and follower Python scripts written for the experiments can be seen from the SV-4 in Figure 16. The leader vehicle script reads the leader's location, including the latitude, longitude, heading, and altitude, and sends the location to the follower vehicle script. A sleep time, or delay, exists in the leader vehicle script that controls how often commands are executed in the script. Sleep time is a parameter used in this research, and will be discussed later in this chapter.

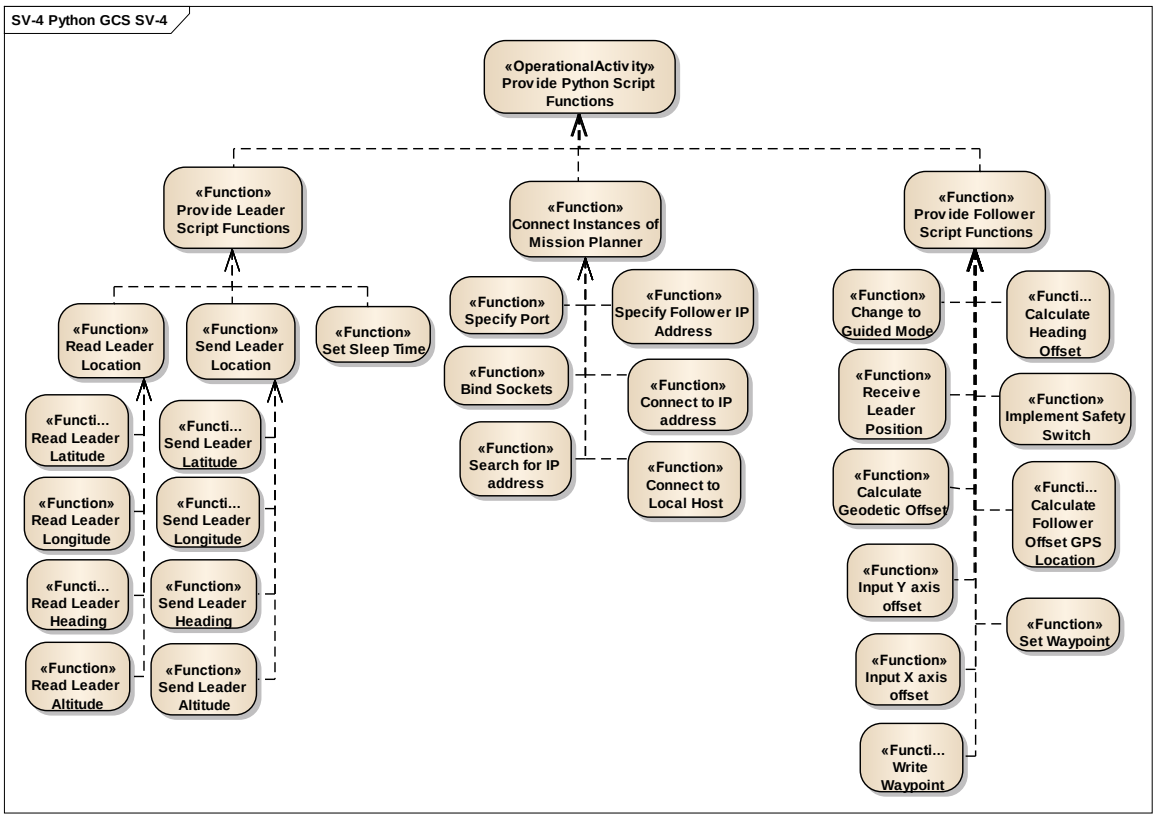

Figure 16. Python System Node Functions SV-4

Instances of Mission Planner can be connected with Python scripts through the use of the follower vehicle's internet protocol (IP) address or a local host. An IP address is used when the instances of Mission Planner are on separate GCSs. A local host is used when the instances of Mission Planner are on the same GCS. By specifying a port in the scripts and creating/binding sockets, a link can be established between instances of Mission Planner.

There are two types of follower vehicle scripts used in this research. One of the follower vehicle scripts calculated a geodetic offset, while the other calculated a heading offset. Both follower vehicle scripts changed the follower vehicle to Guided mode before receiving the leader vehicle's location coordinates. In both scripts an x axis offset and y axis offset are input. In the heading offset follower script, a safety switch was integrated to kill the script if the operator felt the vehicle or surroundings were in danger. Without this safety switch, the Python script would

continue to override operator controls until the script was manually stopped. Once the offsets are calculated in either follower vehicle script, the GPS location of the follower vehicle's offset waypoint is identified. This waypoint is written before it is set.

The functions of Droid Planner 2 are seen from the SV-4 in Figure 17. The Follow Me function is the primary reason for the application's use in this research. The function makes the follower vehicle follow the GCS. A Fly-to-Here function allows the operator to point to a location on the map of Droid Planner 2 and immediately command the vehicle to travel to that location. This Fly-to-Here function sets the vehicle into Guided mode, which is the required mode for the Follow Me function to work. The Follow Me function is begun and stopped by selecting or deselecting the "Follow" command on the menu. The vehicle maintains the same altitude throughout the Follow Me function that it was at when first switched to the Follow Me function. Throughout the vehicle's connection to Droid Planner 2, a T-log is recorded on the GCS.

Like Mission Planner, Droid Planner 2 can set parameters and manage waypoints by editing, writing, or loading previous waypoints from a vehicle to the GCS. Droid Planner 2 displays the location of the vehicles and the GCS by connecting the GCS to the internet and enabling GPS. Once a pair of telemetry modems are connected from the GCS to the vehicle, a connection to Droid Planner 2 from the GCS can be made.

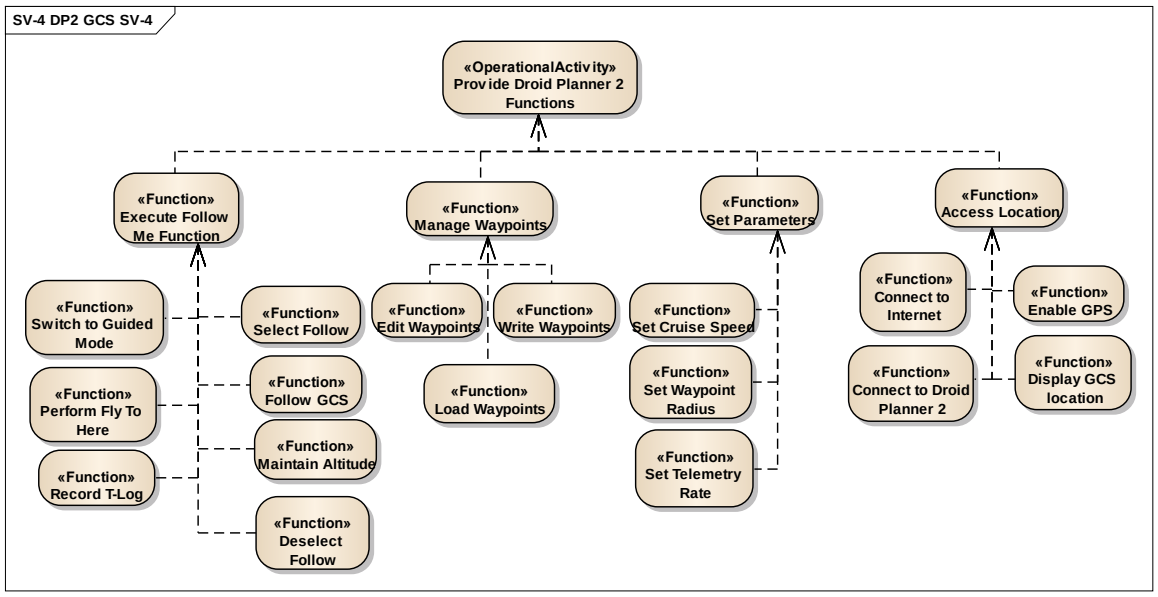

Figure 17. Droid Planner 2 System Node Functions SV-4

The operator functions can be seen from the SV-4 in Figure 18. These functions primarily involve collecting the measures of performance from the experiments in this research. These measures of performance include latency, position accuracy, and figure eight position accuracy, which will be discussed in subsequent sections of this chapter. When in Manual mode, the operator has full control of the vehicle from the radio. From the radio, the operator can switch vehicle modes. The operator can also edit waypoints and input parameter settings on the GCS.

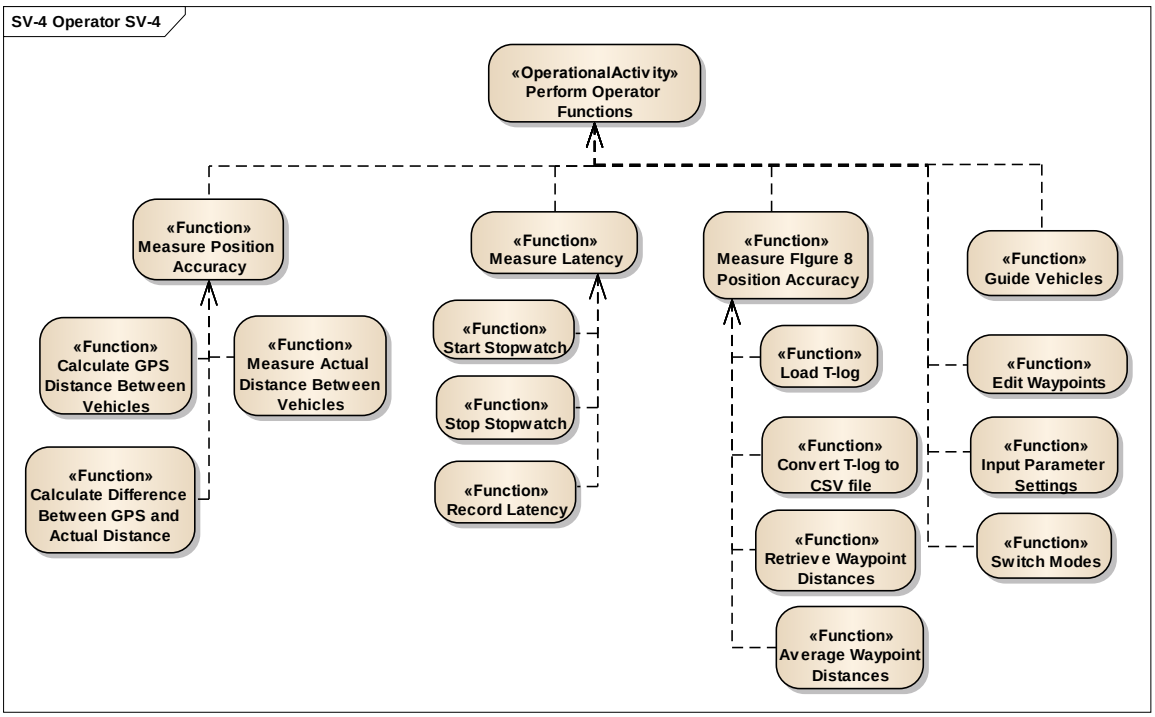

Figure 18. Operator System Node Functions SV-4

Procedures

The architectures and hardware descriptions are used for the development of a set of test procedures that address the investigative questions in this research. First, a baseline cooperative behavior and control method will be established for comparison tests. Mission Planner offers this baseline through a swarm application that is essentially a follow-me method allowing multiple vehicle connections. A new cooperative behavior and control method with similar effects will then be created with Python in order to test how well the two methods on the ground vehicles compare in terms of latency and accuracy error metrics. After finding the best Python configuration, new cooperative behaviors will be introduced into the Python script for Design of Experiments (DOE). DOE will be used to find the best configuration of script sleep time, telemetry modem's max window, position telemetry rate, waypoint radius, and cruise speed settings on Mission Planner. This optimum configuration will be used to determine the best

offset accuracy error between the leader and follower. Then the optimum settings will be used in heterogeneous vehicle configurations between a rover and a multi-rotor to measure the latency and accuracy error. First, the rover will be designated the follower vehicle, and the multi-rotor as the leader vehicle. Then the roles will be switched. Droid Planner 2, a new GCS smart phone application platform will also be used as an experimental method for comparison. Latency and accuracy error will be recorded using optimum settings again for analysis. Through these experiments, latency and accuracy error data will be analyzed through models, spreadsheets, and graphs, to determine the effects and best configuration for the responses.

Mission Planner Swarm

Mission Planner offers a swarm application that allows the user to simultaneously connect to multiple vehicles. This application was briefly discussed in chapter two. The application connects to multiple vehicles through the use of a "Connect to Mavs" button. Once all communication modems to the vehicles are physically connected to the GCS, the leader vehicle is connected and set as the leader. Then the "Connect to Mavs" button is pressed to connect to the other vehicles. The vehicles will then appear at the origin of a grid, where each block represents roughly $1m^2$. Documentation was not found for the area, or distances, of each cube in the offset grid. As observed during the execution of several experiments, the area of each cube in the offset grid appeared to be approximately 1 m^2. The top of the grid is true north, while the right part of the grid is east. The leader vehicle remains stationary in the center of the grid, while the remaining vehicles are set to an offset in relation to the leader by placing them at a desired position away from the leader on the grid. Because the grid is representative of north, south, east, and west, the follower vehicle(s) will maintain position in the formation based on a geodetic frame. In other words, if the follower vehicle(s) are offset to be south of the leader on the grid, the follower(s) will always be south of the leader, regardless of the heading of the leader vehicle. Once the desired offset is established, the "start" command is given. This "start"

command gives continuous waypoint updates to the follower(s) in Guided mode to maintain the established offset in relation to the leader, whether the leader is following defined waypoints, or operated manually. The follower vehicles remain in Guided mode at all times when the swarm application is operating. The navigation of all vehicles will be displayed on the map in Mission Planner during the application run. However, the T-log files, which record the vehicle path and parameters, can be recorded only for the leader vehicle. Log files, containing the same parameters as in the T-log files, are recorded on all Pixhawk autopilots. These Log files can be downloaded from the autopilots. This application will be used as a baseline for comparison against new cooperative behavior and control methods because it is the only known swarm capability associated with Mission Planner or the stock Pixhawk autopilot.

Python

Python is a high-level programming language for general use. It can accomplish the same tasks as other programming languages, such as C; however, it can do it in fewer lines of code. Python can import libraries that provide functions used for writing lines of code to accomplish a task. One of these libraries is Mission Planner. Mission Planner can use Python to give the user flexibility and control over operations and commands in Mission Planner and with connected vehicles. By scripting lines of code and running the script through Mission Planner, waypoint and offset creation can be supported.

Python offers the capability of linking two or more instances of Mission Planner through a network. Therefore, Mission Planner can either communicate to an instance of Mission Planner on another computer or on the same computer. This send and receive ability allows for each Mission Planner instance to connect to a single vehicle and still communicate with each other for cooperative behavior and control. This multiple GCS configuration is vital in situations where more than one aerial vehicle cannot operate from a single ground station, as limited by the Military Flight Release (MFR) conditions.

Through this communicative behavior, the Mission Planner library from Python can create waypoints for the vehicle and set the vehicle into Auto, Guided, or Manual mode. This Mission Planner library is key for use of a follower vehicle in a swarm or flocking configuration. It allows the follower vehicle to read the current GPS coordinates and parameters of the leader vehicle from Mission Planner and add an offset using a meters per latitude calculation. This offset, in return, will give actual GPS coordinates and parameters that can be set as waypoints through Python to Mission Planner for the follower vehicle to read. With the use of a loop, the vehicle parameters can be read continuously from the leader and create waypoints for the follower vehicle to follow. Based on the leader's position, the offset will always give a desired location for the follower vehicle, resulting in a leader-follower configuration. Mission Planner and Python will be used to script the same cooperative behavior, as well as new behaviors, to compare to and improve upon Mission Planner's swarm application.

Test 1: Configuration Comparison

Once a Python script is created to simulate Mission Planner's swarm application, the performance of the configuration will need to be determined. While Mission Planner's swarm application must be run from a single computer, Python offers the versatility to demonstrate cooperative behavior and control from multiple computers. Using rover ground vehicles, the follower vehicle can operate from a different computer than the leader vehicle. Little is known about whether this configuration is more effective than operating from a single computer or how it compares to Mission Planner's swarm application. Hence, the two different configurations, depicted in Figures 11 and 12 will be compared with each other as well as the baseline architecture depicted in Figure 10.

Performance metrics are defined here as latency, in seconds, and accuracy error, in meters. Though accuracy error measurements are recorded in inches, the response values use are converted to meters for consistency with other measurements. Low latency is desired and is

critical in mission operations. Cooperative behavior and control requires low latency in order to speed up response times and communicate instructions across platforms in time-constrained missions. Latency will be measured by starting a stop watch when the leader vehicle takes off and stopping the stop watch once the follower vehicle responds. The time in seconds will be recorded for five runs or trials per configuration, allowing an average and standard deviation to be obtained for each configuration. Accuracy error is vital in missions where targets must be identified, tracked or neutralized. Cooperative behavior and control often requires low accuracy error to avoid collisions with other vehicles. Similar to the challenges associated with latency, the more vehicles involved with cooperative behavior and control, the more vital a low accuracy error is. A lag in instruction could also incur position inaccuracies, because each vehicle is basing its own movement on the observance of associated vehicles. The accuracy error involves calculating the theoretical distance in inches by recording the GPS coordinates of the leader and follower vehicles from Mission Planner and finding the difference. Then the actual distance from the leader and follower vehicles will be physically measured and subtracted from the system's estimated distance, calculated between the two GPS coordinates. The absolute value of the differences between these distances will give the error in inches. Five runs or data points will be collected for each configuration, culminating in an average and standard deviation for each configuration. This average will then be converted to meters. However, accuracy error will not be measured with this accuracy measurement method using Mission Planner's swarm application because the application allows only one instance of Mission Planner for both vehicles. This allows the ability to retrieve the GPS coordinates through Mission Planner for the leader vehicle only. By measuring the difference between actual distance and commanded offset, the Mission Planner swarm application's accuracy error can be measured.

Test 2: Optimum Factor Settings (Design of Experiments)

Proving that multiple vehicles can operate with the two methods and finding the most efficient Python method configuration will pave the way for improvement. Once the Python method has been demonstrated against Mission Planner's swarm application, the Python script will be modified to incorporate new behaviors. The script will include Equation 8 and Equation 9 to orient the follower vehicles' position based on the leader vehicle's heading instead of geodetic location. This can be done by rotating the frame based on the heading angle of the leader and adding the offset to the leader position. Follower Latitude position is the latitude coordinate for the follower vehicle and Follower Longitude position is the longitude coordinate for the follower vehicle. The leader latitude and longitude coordinates are represented by Lat_{leader} and Lng_{leader}. The latitude and longitude offset, Lat_{offset} and Lng_{offset}, are calculated with the input of a desired x and y value offset, in meters, for the follower from the leader divided by meters per degree. The heading angle of the leader vehicle is represented by θ.

$$Follower\ Latitude\ position = Lat_{leader} + Lng_{offset} * \sin\theta + Lat_{offset} * \cos\theta \quad \textbf{(8)}$$

$$Follower\ Longitude\ position = Lng_{leader} - Lng_{offset} * \cos\theta + Lat_{offset} * \sin\theta \quad \textbf{(9)}$$

The addition of a safety switch functionality will be added to the Python script as well. Normally, when running the Python script or Mission Planner Swarm application, the safety pilot's manual radio controls are overridden. This introduces a safety hazard in many circumstances if control cannot be given back to the operator fast enough. By observing the mode switch input channel to the autopilot, the power level of the channel is an indicator of the commanded operating mode. When the vehicle is on and connected to Mission Planner, the manual, auto, and stabilize switches can be switched on using the safety pilot's radio, and the power levels are seen in the vehicle's telemetry in the Flight Data tab of Mission Planner. Each switch will trigger different power levels for that channel number parameter. Each type of remote

control radio will not necessarily have the same power levels for each switch so the switching levels must be chosen specific to the radio being used. Once the set power levels for the specified vehicle are found for each switch, an "if" statement can be programmed into the Python script to trigger a shutdown of the script using the channel number parameter power levels once the appropriate switch is triggered.

Once the new behaviors are incorporated into the Python script, Design of Experiments (DOE) will be used to find sets of optimum factor levels for latency and accuracy error using two rover ground vehicles, one as the leader and the other as the follower. First, the experiments will be performed with accuracy error as the response and then with latency as the response. A low accuracy error is desired because it is measured by the distance difference in inches between the actual and calculated GPS distance of the vehicles. A low latency is also desired because a lower latency time should allow tighter formations or following applications. Using DOE, two level factors will be investigated with the two methods to determine which factors are significant. The factors will only have two levels, high and low values, to create simpler linear models and lower the amount of runs needed. The high and low points of the factors are only able to create a linear prediction. Finding these significant factors will facilitate a model using JMP Pro 11, a statistical software package [31]. This model will give the optimum values, high or low, for each significant factor in order to get the most desirable response. The linear model can also be tested through a lack-of-fit test to determine if a higher order model is needed.

Following the latency and accuracy error experiments, another accuracy error measurement method will be executed. A path consisting of figure eight waypoints will be assigned to the leader rover vehicle. This will demonstrate how the follower vehicle responds to the turns and maneuvering from the leader vehicle once the Python script is run. The T-log of the follower vehicle will capture the waypoint distance for the follower vehicle at the telemetry rate setting. These intervals capture two or three data points each second which are then averaged

into one single data point for every run of factor settings in the sixteen run design. The standard deviation will be calculated as well. Each run will last about one to two minutes.

When running the Python script, the follower vehicle remains in Guided mode with the guide to waypoint updated every time the leader vehicle's location is updated. Therefore, the waypoint distance measures how far away the follower vehicle is away from the last known position of the leader vehicle, in meters, at all times unless there is a follower vehicle offset given. There will be a waypoint lag in the measurements due to the latency of the follower vehicle waypoint updates, given from the Python script. Still, the accuracy error measurement should give a good understanding of how close to the target the follower vehicle is actually following the leader.

This measurement method will be affected by latency since the waypoints are updated only as fast as this latency allows. The measurement method gives an alternative way of measuring accuracy error. Specifically, it includes both vehicles in motion, while the previous accuracy error measurement method consists of stationary vehicle measurements. Both methods will be analyzed and used for comparative purposes to determine which accuracy error measurement method is better.

Factors

The factors chosen to be tested are the position telemetry rate, waypoint radius, vehicle cruise speed, leader sleep time, and the 3DR modem's max window and are seen in Table 4. As a subject matter expert, these factors were chosen based on an exhaustive search for latency and position accuracy sources. A factor's level can be represented in its actual units or coded units. When using DOE, the coded units line up the regression model's intercept to the center of the design, whereas the actual units usually have intercepts far from the design space. The coded units also eliminate units of measure allowing similar coded unit levels to hold the same weight between factors. For a two level model, a high and low level are represented by a +1 and a -1

coded unit. Center runs have a 0 coded unit, as the value in between the high and low levels. The first three factors can be set in Mission Planner under the "Config/Tuning" tab.

The position telemetry rate defines how often the position telemetry data is updated from the vehicle to Mission Planner and can be set from zero to ten, but the default is three. Ten is the fastest setting, while zero is the slowest. Therefore, ten will be set as the high value, or positive one, and three will be set as the low value, or negative one.

Table 4. DOE Factor Levels

Factor	Low Level (-1)	High Level (+1)
Telemetry Rate (Hz)	3	10
Waypoint Radius (m)	0.25	5
Cruise Speed (m/s)	1	6
Sleep Time (ms)	500	5000
Max Window (ms)	33	131

The waypoint radius controls the radius of entry into the waypoint. Once the vehicle lands within the radius of that waypoint, it confirms it has arrived at the waypoint and begins moving towards the next waypoint. The waypoint radius is in meters and gives a text box for setting entry. Therefore, any value can be set in it, making it a continuous factor. However, in terms of leader-follower, setting the follower waypoint radius to zero will force the follower vehicle to crash into the leader vehicle. Hence, 0.25, instead of zero, will be used as the low value, or negative one. The high value will be set to five because five meters tends to be around average GPS error. No offset was used for the tests.

The vehicle cruise speed setting controls how fast, in meters per second, the vehicle travels in Auto mode when navigating across waypoints. Once again, the value is placed in a text box entry. For this experiment, the low value will be set to one and the high value set to six m/s.

The leader sleep time is built into the Python script. This sleep time determines how often, in milliseconds, the leader vehicle's telemetry data from Mission Planner is sent to the follower vehicle's Mission Planner Python script. Therefore, this factor will be used only for the

Python method instead of the Mission Planner swarm application method. The low value will be set to the smallest, or fastest, stable sleep time, before the script starts sending repetitive position data, which was found to be about 0.5 seconds or 500 milliseconds. The high value will be set to five seconds or 5000 milliseconds because higher times start to induce much larger latencies.

The 3DR modem's Max Window is used to set how often the GCS sends a packet to the vehicle, in milliseconds. The default, which is why it was chosen as the high value, for this factor will be set to 131 and the low value to 33 because it is the lowest setting possible. Therefore, when the factor is set to the high value, the GCS will send a packet to the vehicle every 131 msecs. Both the modem connected to the GCS, and to the vehicle must always contain the Max Window to be able to communicate with one another [23].

Since there will be five two level factors for the Python method, there will be sixteen experimental runs or data points with different treatments, plus four center runs, totaling twenty runs, seen from Table 5. Though the design in Table 5 is not randomized in order to display the runs in an organized manner, the design will be randomized when executed. This randomization is to avoid the effect of unknown nuisance factors with the experimental factors. A center run will be the first and last runs in the experiment with the other two center runs spaced equally apart between runs in order to provide a measure of stability. The number of runs is found by taking 2^{5-1} and adding the four center runs. This 2^{5-1}, with five factors, is called a fractional factorial design [32]. These designs are created to limit the amount of runs and still produce effective predictions through the regression model. If a standard full factorial design were to be used with five factors, thirty-two runs would be required, which could waste time and resources when a fractional factorial could produce a similar sufficient model. However, not all fractional factorial models are equally effective. The less runs in a high factor populated model, the more bias is introduced into the model. For instance, super saturated designs have less runs than factors which prevents main effects from being estimated. Therefore, high resolution designs are desired so that

factor main effects aren't aliased with other significant factors or interactions. When aliased, it isn't evident which aliased factor triggers a recognized effect. Therefore, a resolution V design will be made for the 2^{5-1} fractional factorial because it is the highest resolution design for the fractional factorial.

Table 5. Test 2 Five Factor Half-Fractional Factorial Design

	Factor Levels				
Run	Waypoint Radius	Cruise Speed	Sleep Time	Max Window	Telemetry Rate
1	-1	-1	-1	-1	1
2	1	-1	-1	-1	-1
3	-1	1	-1	-1	-1
4	1	1	-1	-1	1
5	-1	-1	1	-1	-1
6	1	-1	1	-1	1
7	-1	1	1	-1	1
8	1	1	1	-1	-1
9	-1	-1	-1	1	-1
10	1	-1	-1	1	1
11	-1	1	-1	1	1
12	1	1	-1	1	-1
13	-1	-1	1	1	1
14	1	-1	1	1	-1
15	-1	1	1	1	-1
16	1	1	1	1	1
17	0	0	0	0	0
18	0	0	0	0	0
19	0	0	0	0	0
20	0	0	0	0	0

The center runs are used as a coded value of zero for the factor levels. Therefore, the factor levels equal distance from the high and low factor levels are used as the center run settings. The center runs will help test for curvature in the model. For each treatment in the model, an average run will be taken out of three repeated runs. These replications are to aid in the confidence, or precision of the estimate for the mean response at each factor level combination, of the data collection. However, the figure eight accuracy error measurements will only be executed once per treatment, averaging the waypoint distances gathered from the T-log.

Test 3: Rover Ground Vehicle Following Multi-Rotor

Once optimum factor levels are obtained for low latency and accuracy error, they will be used in the differing heterogeneous vehicle configurations. The accuracy error measured from the averages of the waypoint distances captured by the T-logs will be used when multi-rotor vehicles are being tested due to its simplicity. The first heterogeneous vehicle configuration used will be with the multi-rotor vehicle set as the leader and the rover vehicle set as the follower vehicle.

Using the Python script with the two vehicle configuration, the same latency and accuracy error tests will be used as from test 2. However, only the optimum factor levels will be used in the run for each response. Again, the latency test will be run three times with the optimum factor levels, averaged into one data point. The accuracy error will be measured by averaging the waypoint distances of the follower vehicle's T-log after the accuracy error test with its optimum factor levels. The standard deviations will be collected as well.

Test 4: Multi-Rotor Following Rover Ground Vehicle

The next heterogeneous vehicle configuration testing will involve the multi-rotor following the rover ground vehicle. That is, the multi-rotor set as the follower vehicle and the rover vehicle set as the leader vehicle. The same latency and accuracy error tests with the Python script will be executed as in test 3, using optimum factor levels. However, the Droid Planner 2 application's Follow Me Mode will be used with the latency and accuracy error tests as well, to provide another method of comparison to the Python method. This method's architecture was seen in Figure 13. The application will be used on a Samsung Galaxy S3 smart phone. The phone will be strapped down on top of the rover vehicle, essentially allowing the multi-rotor to follow the phone when in Follow Me Mode.

Analysis

Once all data is gathered, trade studies associated with camera settings and vehicle operating parameters will be performed. The camera will be presumed attached to the multi-rotor as a follower and the rover ground vehicle as the leader vehicle. Using the collected data, a set of camera Field of View (FOV) angles and altitude will be altered to reflect the largest time buffer for the ground vehicle to travel outside the camera's footprint, with the multi-rotor stationary. This time buffer will involve the ratio between the time it takes for the rover to exit the footprint, and the latency of the follower vehicle. The speed of the rover and track lag of the follower vehicle will be used based on the data collected from the previous experiments.

Summary

Using ground control stations, autopilots, and a combination of aerial and ground vehicles, a set of procedures was developed in order to address the research's investigative questions. Two methods are defined using Python with Mission Planner and Mission Planner's swarm application. Latency and accuracy error are defined as the cooperative behavior and control metrics for this research due to their mission criticality for autonomy. Latency should be minimized for instructions to be passed quickly across vehicles and reduce response times, while accuracy error could prevent vehicle collisions and mission obstruction. These methods and metrics must first be tested with ground vehicles due to their safe nature in comparison to aerial vehicles and the challenges associated with flight policy. Using Python with Mission Planner from separate computers for each vehicle and still exhibiting cooperative behavior and control, presents an opportunity to find out which platform configuration demonstrates lower latency and more accurate positioning. The most effective Python solution will be compared to the Mission Planner swarm application's outcomes as a baseline test.

With the aid of statistical software, DOE will allow for position telemetry rate, waypoint radius, vehicle cruise speed, leader sleep time, and 3DR modem's max window to be used as factors in testing of the two methods. The DOE tests will include latency as the response and accuracy error as the response for a rover-following-rover vehicle configuration. Accuracy error will also be measured in terms the average waypoint distances captured from a T-log for the follower vehicle post-operation. A 2^{5-1} Resolution V fractional factorial sixteen run design will be used with four center runs. The Python script used for the DOE will involve new behaviors such as a follower vehicle offset in relation to the heading of the leader vehicle, and a safety switch for precaution.

Once the methods are proven on ground vehicles, they can be implemented with aerial multi-rotors and heterogeneous vehicle configurations. However, only the Python method will be able to be run with the multi-rotors as followers due to safety concerns. The performance of these multi-rotors will lead to heterogeneous vehicle testing, with the multi-rotor as the leader first and the ground vehicles as the followers. The optimum factor levels for a low latency and low accuracy error, using the figure eight measurement method, will be used to find the latency and accuracy error for the vehicle configuration. This will exhibit the first confirmation of the DOE models created.

The heterogeneous vehicle configuration will then be reversed, with the rover ground vehicle as the leader and the multi-rotor as the follower vehicle. The same tests with latency and accuracy error as the last heterogeneous vehicle configuration test will be done with the DOE optimum factor levels. These optimum factor levels will also be run on a Droid Planner 2 application via smart phone to measure latency and accuracy error through the use of the application's Follow Me mode.

The consistency of the factors, metrics, and methods set up the data for comparative analysis. Therefore, safe assumptions can be made when there are noticeable differences in

effects based on vehicle configuration. These vehicle configuration effects lead to application use that will be demonstrated through trade studies using camera FOV, altitude, speed, and latency of the vehicles. Based on these parameters, the relationship between them will be determined as well as a chance to see how long the rover can be kept within the camera's footprint.

IV. Analysis and Results

Chapter Overview

The analysis and results chapter discusses the results obtained from the implementation of the previous chapter's methodology. The chapter covers the results of diagnostic testing, using the same parameters on Mission Planner's Swarm application as with the Python method, in order to compare the latency and accuracy error between a leader and follower rover vehicle. Using the Python method, Design of Experiments was used to find the optimum parameter settings for both latency and accuracy error responses. These optimum parameter settings were used on heterogeneous vehicles, ground rover vehicle and multi-rotor, using the Python method and Droid Planner 2 to compare the effects of latency and accuracy error on each configuration. These results are broken down and analyzed in order to decipher the relationship between the data and findings.

Diagnostic Testing

Using Mission Planner's Swarm application and Python, latency in seconds and accuracy error in inches were directly compared between both methods. The Python method consisted of two different configurations, one with one instance of Mission Planner running on two different computers, and one with two instances of Mission Planner running on the same computer. The same parameters were used on all methods and configurations for comparative purposes, with sleep time set at 1000, or one second. Five data points were taken for each method. The average of the five data points and their standard deviation for each method are seen below in Table 6.

The latency was measured using a stopwatch. The stopwatch was started when the leader vehicle took off, and stopped when the follower vehicle physically responded to the leader's location. The accuracy error was measured by taking the GPS coordinates of the leader and follower vehicles from Mission Planner, after the follower vehicle reached its appropriate offset

from the leader. The actual distance was then measured between the two vehicles with measuring tape. The offset was then calculated between the two GPS points and compared to the actual distances measured. The difference between the actual distance and the calculated GPS points distance is considered as the accuracy error.

Table 6. Diagnostic Testing for MP Swarm and Python Configurations

	Latency (sec)	Std. Dev.	Accuracy Error (m)	Std. Dev.
MP Swarm	2.67	0.81	N/A	N/A
1 MP on each (2) PCs	2.99	0.89	0.77	0.41
2 MP on 1 PC	4.47	1.42	0.76	0.49

Accuracy error is not shown for Mission Planner's Swarm application as it did not have an accuracy error. This is due to Mission Planner only being able to read parameters, and save T-logs, off of one vehicle at a time. In this case, the follower vehicle's GPS location could not be read through Mission Planner. Accuracy error could not be measured by comparing the actual distance to the theoretical offset because the offset scale from the grid of the Mission Planner Swarm application was unknown. Documentation was not found for the area, or distances, of each cube in the offset grid. Through the execution of a couple of experiments, the area of each cube in the offset grid appeared to be closest to 1 m^2.

The latency averages were then compared between the two Python configurations to test if one mean was larger than the other. Assuming the variances of each Python configuration are not equal, the two-sample t-test, t_0, seen from Equation 10 was used to test whether the separate GCSs Python configuration's mean latency, $\bar{y}_1$, was the same as the single GCS Python configuration's mean latency, $\bar{y}_2$. Equal means between the Python configurations represented the null hypothesis. The two-sample t-test also tested if the single GCS Python configuration had a higher latency mean than the separate GCSs Python configuration's latency mean, which was the alternative hypothesis [32]. The sample variances for the single GCS Python configuration,

S_1, and the separate GCSs Python configuration, S_2, are each squared and divided by sample sizes of the single GCS Python configuration, n_1, and the separate GCSs Python configuration, n_2. The degrees of freedom, v, were found from Equation 11.

$$t_0 = \frac{\bar{y}_1 - \bar{y}_2}{\sqrt{\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}}} \tag{10}$$

$$v = \frac{\left(\frac{S_1^2}{n_1} + \frac{S_2^2}{n_2}\right)^2}{\frac{(S_1^2/n_1)^2}{n_1 - 1} + \frac{(S_2^2/2)^2}{n_2 - 1}} \tag{11}$$

The results in Table 7 show a p-value of 0.048, which is less than an α of 0.05. This low p-value shows that the null hypothesis is rejected and that operating the vehicles from the same GCS using Python scripts produces a significantly higher latency than operating the vehicles from separate GCSs using Python scripts.

Table 7. Two-Sample T-Test Results Between Python Configurations

t_0	v	p-value
1.98	6.71	0.048

A fact to note here is that using the Python method with two instances of Mission Planner on a single computer, the latency increased 1.5 times than that of using one instance of Mission Planner on each computer. This could be attributed to computer processor speed or the two modems competing for processor time. There could be increased lag with two applications up and working on the same computer, apparently more so than lag across wireless network connections. Having two modems connected to a single computer's USB port could require higher processing capabilities. Looking at the latency standard deviation, the two instances of Mission Planner on one computer has a noticeably higher standard deviation than the other Python configuration and Mission Planner Swarm. This could be due to some erratic data points (outliers) involved with the collection, or perhaps the configuration results are just not

predictable. Either way, the standard deviation still was not high enough to ignore the Python configurations' effect on latency. There was also little difference between both Python configurations' accuracy error. Therefore, for implementation purposes, the Python method used for continuing experimentation was one instance of Mission Planner for each computer.

Optimum Factor Settings (Design of Experiments)

Once the preferred Python method was recognized, the optimum parameter settings were chosen for telemetry rate, waypoint radius, cruise speed, sleep time, and max window, seen from Table 4. Using Design of Experiments with the five factors, a resolution V 2^{5-1} fractional factorial design was created. This created a sixteen run design; however, two extra replicates of each run were executed to improve the estimate of the mean at each design point. These replicates did not affect the size of the design, in an analysis sense, because the response averages of the replicates for each parameter settings were taken as the single response for each run in the original 16-run design. The same designs were used for both responses, latency and accuracy error. However, the latency design was executed with four center runs because curvature was suspected in the model. The accuracy error model did not contain center runs due to the lack of a credible model or way to measure the accuracy error. The design was created and data analyzed through JMP 11 Pro.

Latency

For the latency model, the data was screened for possible significant effects, as seen in Figure 19.

Contrasts

Term	Contrast	Lenth t-Ratio	Individual p-Value	Simultaneous p-Value	Aliases
Sleep Time	2.93992	14.99	<.0001*	0.0003*	Max Window*Cruise Speed*Telemetry Rate*WP Radius
Max Window	0.71090	3.62	0.0069*	0.0817	Sleep Time*Cruise Speed*Telemetry Rate*WP Radius
Cruise Speed	0.18320	0.93	0.3336	0.9998	Sleep Time*Max Window*Telemetry Rate*WP Radius
Telemetry Rate	-0.14927	-0.76	0.4296	1.0000	Sleep Time*Max Window*Cruise Speed*WP Radius
WP Radius	0.14774	0.75	0.4336	1.0000	Sleep Time*Max Window*Cruise Speed*Telemetry Rate
Sleep Time*Sleep Time	1.00128	5.10	0.0017*	0.0173*	Max Window*Max Window, Cruise Speed*Cruise Speed, Telemetry Rate*Telemetry Rate, WP Radius*WP Radius
Sleep Time*Max Window	-0.47756	-2.43	0.0310*	0.3060	Cruise Speed*Telemetry Rate*WP Radius
Sleep Time*Cruise Speed	-0.25770	-1.31	0.1875	0.9527	Max Window*Telemetry Rate*WP Radius
Max Window*Cruise Speed	-0.07622	-0.39	0.7209	1.0000	Sleep Time*Telemetry Rate*WP Radius
Sleep Time*Telemetry Rate	-0.05052	-0.26	0.8078	1.0000	Max Window*Cruise Speed*WP Radius
Max Window*Telemetry Rate	-0.98486	-5.02	0.0020*	0.0186*	Sleep Time*Cruise Speed*WP Radius
Cruise Speed*Telemetry Rate	0.16679	0.85	0.3743	1.0000	Sleep Time*Max Window*WP Radius
Sleep Time*WP Radius	0.11383	0.58	0.5948	1.0000	Max Window*Cruise Speed*Telemetry Rate
Max Window*WP Radius	0.02555	0.13	0.9001	1.0000	Sleep Time*Cruise Speed*Telemetry Rate
Cruise Speed*WP Radius	-0.67665	-3.45	0.0086*	0.0976	Sleep Time*Max Window*Telemetry Rate
Telemetry Rate*WP Radius	-0.51338	-2.62	0.0250*	0.2626	Sleep Time*Max Window*Cruise Speed
Null18	-0.02376	-0.12	0.9067	1.0000	
Null19	-0.09245	-0.47	0.6622	1.0000	
Null20	-0.09951	-0.51	0.6413	1.0000	

Figure 19. Latency Model Screening

As seen in Figure 19, Sleep Time and Max Window appear to be the significant main effects, because they have p-values lower than 0.05. This can also be seen from the Half Normal Plot in Figure 20. Those two main effects are the farthest from the blue line, which signifies irrelevance. Sleep Time*Max Window, Max Window*Telemetry Rate, Cruise Speed*WP Radius, and Telemetry Rate*WP Radius also appear to be significant.

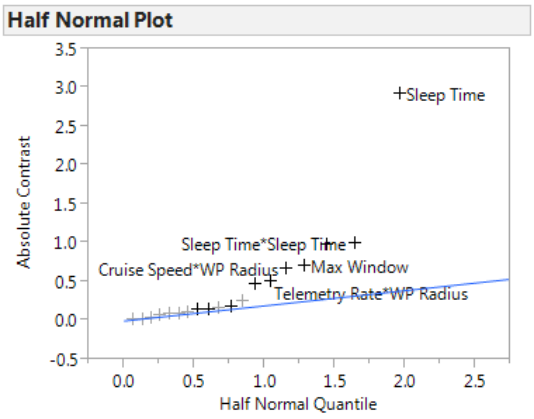

Figure 20. Latency Half Normal Plot

The effects tests are seen in Figure 21. Sleep Time was found to be a direct reflection on latency and was, therefore, very significant. Sleep Time was placed in the leader's Python script to specify time between the leader vehicle's position data being sent to the follower vehicle's GCS. If no Sleep Time is specified in the code, the default is 1000 ms, or 1 second. As the Sleep Time was increased it was very apparent that the latency increased exponentially. Therefore, a

lower Sleep Time is desired. Max Window was a setting for the 3DR modems that controlled how often the vehicle sent a packet to the GCS in milliseconds. Logically, a lower Max Window should be desired. This would affect how often the position information is received and reaction time of the follower vehicle. Both radios on the channel had to have the same Max Window or they couldn't communicate with one another.

Notice how Sleep Time*Sleep Time, or $(\text{Sleep Time})^2$, is in the model and is even proven significant. Normally, a three or higher level design is needed to estimate individual quadratic factors. However, this term tests for curvature in the model because it is the only quadratic term shown in the two level design. Center points are not considered as a third level. A three level design was not created in the interest of time and due to the two level design having such a high $R^2_{adjusted}$. Therefore, the term's significance shows that curvature does exist in the model.

Effect Tests

Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
Sleep Time	1	1	172.86267	575.2806	<.0001*
Max Window	1	1	10.10747	33.6373	0.0003*
Cruise Speed	1	1	0.67121	2.2338	0.1692
Telemetry Rate	1	1	0.44566	1.4831	0.2542
WP Radius	1	1	0.43656	1.4528	0.2588
Sleep Time*Sleep Time	1	1	20.05113	66.7294	<.0001*
Sleep Time*Max Window	1	1	4.56132	15.1799	0.0036*
Max Window*Telemetry Rate	1	1	19.39896	64.5590	<.0001*
Cruise Speed*WP Radius	1	1	9.15713	30.4746	0.0004*
Telemetry Rate*WP Radius	1	1	5.27127	17.5426	0.0023*

Figure 21. Latency Effect Tests

The other main effects are included in the model, despite being insignificant, for hierarchy because there are interactions in the model with these factors that proved to be significant. Significant interactions show that the interaction between two factors have a significant effect on the response and that the effect of one factor depends on the factor level of another factor.

Figure 22 shows an $R^2_{adjusted}$ of 0.97676. Note that a perfect fit to the data gives an $R^2_{adjusted}$ of 1. This shows that the model fits the data extremely well. The model's significance is also proven in Figure 23 with a p-value much lower than 0.05.

Summary of Fit

RSquare	0.988992
RSquare Adj	0.97676
Root Mean Square Error	0.548164
Mean of Response	6.527555
Observations (or Sum Wgts)	20

Figure 22. Latency Summary of Fit

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	10	242.96338	24.2963	80.8573
Error	9	2.70436	0.3005	Prob > F
C. Total	19	245.66774		<.0001*

Figure 23. Latency ANOVA Table

The Residual vs Predicted Plot in Figure 24 shows a fairly constant variance throughout the graph. The normality plot in Figure 25 shows that there is no issue with normality because all data points seem to fall along the linear line. Therefore, these two graphs show that the model is adequate and no transformations should be needed.

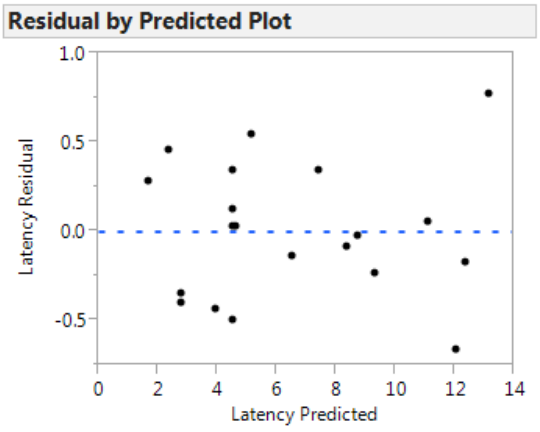

Figure 24. Latency Residual vs. Predicted Plot

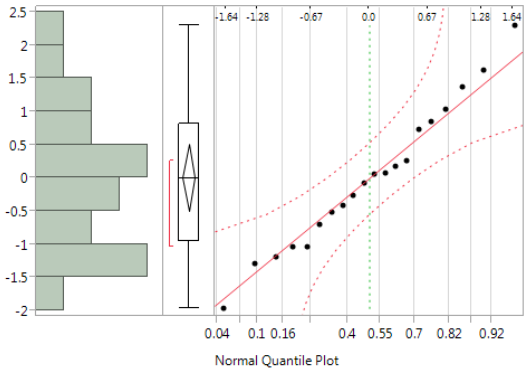

Figure 25. Latency Normality Plot

There appears to be no lack of fit in the model because the lack of fit test in Figure 26 shows a p-value above 0.05. The lack of fit tests how well the model fits the data. For example, if a linear model exhibits lack of fit, the factor-to-response relationship will not be characterized properly and a higher order model would be needed [33].

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	6	2.3240567	0.387343	3.0556
Pure Error	3	0.3803000	0.126767	**Prob > F**
Total Error	9	2.7043567		0.1937
				Max RSq
				0.9985

Figure 26. Latency Lack of Fit Test

The model parameter estimates can be seen in Figure 27. A design containing only two level factors limits the model to a linear prediction. Without a third level in a factor, quadratic effects can't be predicted. Though including quadratic terms in a two level model is normally avoided, a case can be made for keeping (Sleep Time)^2 in this two level latency model. Looking at the model without including (Sleep Time)^2 in it, not only does there prove to be lack of fit from Figure 28, but Figure 29 shows that the $R^2_{adjusted}$ has dropped significantly to 0.824009. There appears to be lack of fit in the model without including (Sleep Time)^2 because the (Sleep Time)^2 Sum of Squares is then added to the Lack of Fit Sum of Squares. The non-linearity in the design

appears to be due to Sleep Time's non-linear increase in latency as the factor increases. There is a significant drop in latency from a five second Sleep Time to a half second Sleep Time. The center point response values were not higher or lower than the lowest or highest response values in the design, which shows there wasn't an extreme amount of curvature, and this helps validate leaving (Sleep Time)^2 in the model.

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	4.525	0.274082	16.51	<.0001*
Sleep Time	3.2869312	0.137041	23.99	<.0001*
Max Window	0.7948063	0.137041	5.80	0.0003*
Cruise Speed	0.2048187	0.137041	1.49	0.1692
Telemetry Rate	-0.166894	0.137041	-1.22	0.2542
WP Radius	0.1651813	0.137041	1.21	0.2588
Sleep Time*Sleep Time	2.5031938	0.306433	8.17	<.0001*
Sleep Time*Max Window	-0.533931	0.137041	-3.90	0.0036*
Max Window*Telemetry Rate	-1.101106	0.137041	-8.03	<.0001*
Cruise Speed*WP Radius	-0.756519	0.137041	-5.52	0.0004*
Telemetry Rate*WP Radius	-0.573981	0.137041	-4.19	0.0023*

Figure 27. Latency Parameter Estimates

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	7	22.375189	3.19646	25.2153
Pure Error	3	0.380300	0.12677	Prob > F
Total Error	10	22.755489		0.0114*
				Max RSq
				0.9985

Figure 28. Latency Lack of Fit Test Without Sleep Time^2 In Model

Summary of Fit

RSquare	0.907373
RSquare Adj	0.824009
Root Mean Square Error	1.508492
Mean of Response	6.527555
Observations (or Sum Wgts)	20

Figure 29. Latency Summary of Fit Without Sleep Time^2 In Model

In the end, (Sleep Time)^2 is left in a two level model based on assumptions such as a high $R^2_{adjusted}$ and there being such few significant main effects. In two level designs, a quadratic term in the model is aliased with all quadratic terms. However, if only Sleep Time and Max Window are significant, with Sleep Time being much more significant than Max Window, then it is logical to assume that the sum of the aliased quadratic terms is mostly the Sleep Time because the other effects are so small. Therefore (Sleep Time)^2 remains in the model and is assumed to be a satisfactory estimate of this term.

Normally, two level designs are used in conjunction with follow on testing. Central Composite Designs (CCD) are suitable designs for follow on experimentation. These designs contain axial runs which are a third factor level, allowing models to be able to predict quadratic terms. Unintentionally, follow on experimentation was not performed with this research. By the time it was realized that follow on experimentation should have been executed, it was too late. Tests 3 and 4 had already used the optimum factor settings from the models.

Using the parameter estimates from Figure 27, the regression model is shown in Equation 12. The factors with bigger coefficients, such as Sleep Time, Max Window, and most of the interaction terms, prove to be more significant. The faster that these factors change, the faster the latency response, y, will change.

$$x_1 = WP\ Radius, x_2 = Cruise\ Speed, x_3 = Sleep\ Time, x_4 = Max\ Window, x_5 = Telemetry\ Rate,$$

$$y = Latency$$

$$y = 4.525 + 0.1651813x_1 + 0.2048187x_2 + 3.2869312x_3 + 0.7948063x_4 - 0.166894x_5 - 0.756519x_1x_2 - 0.573981x_1x_5 - 0.533931x_3x_4 + 1.101106x_4x_5 + 2.5031938x_2^2 \quad \textbf{(12)}$$

The prediction profiler in Figure 30 shows what factor settings or levels will give the most optimal response, which in this case is low latency. All factors should be at negative one coded

values, while Sleep Time should be at -0.763198 coded value to get the predicted lowest latency of -0.36256 seconds. Obviously, there is more error in the prediction at these factor settings. However, what this does say is that the lowest latency possible as predicted by the model can be achieved with these settings. This also confirms the regression model in Equation 12. As the factors' values get lower, the latency gets lower as well.

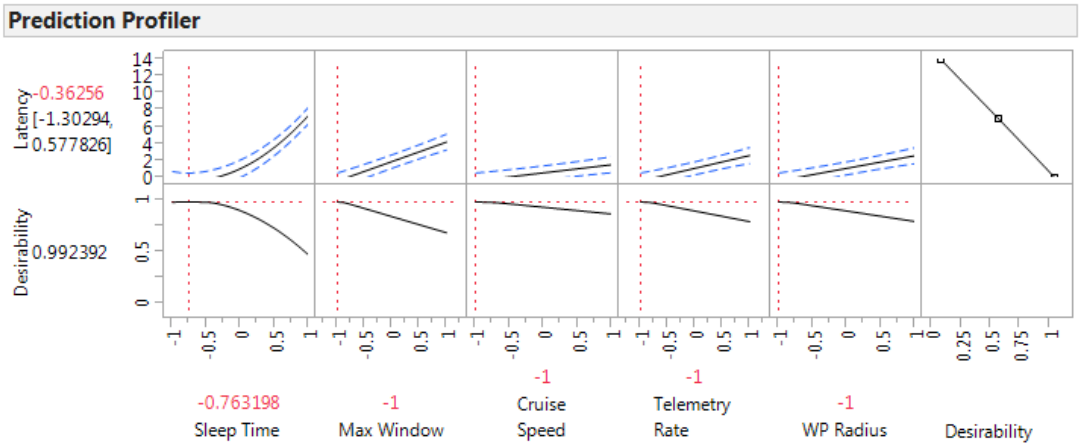

Figure 30. Latency Prediction Profiler

Figure 30 also show how Sleep Time has an exponential effect on latency. This gives further evidence that leaving (Sleep Time)^2 in the two level latency model will still give the same results. For example, a lower Sleep Time will always result in lower latency according to this model. If Sleep Time had a quadratic effect on latency, then the optimum factor level may not be as easy as selecting the lowest or highest factor level.

The prediction profiler is validated through the cube plot in Figure 31. Notice that from the Cube Plot, the corner of the cube with all factor settings set to negative one, or low values, predicts the lowest latency at -0.222 seconds. This is very similar to the Prediction Profiler in Figure 30; however, the experimental run with all these factors set to negative one was never executed in the experiment, which would explain the error from the negative prediction estimate. However, this shows the advantages of fractional factorial designs by being able to predict

response values of factor settings having never run them. These predictions allow for fewer runs and resources to be used in fractional factorial designs rather than full factorial designs.

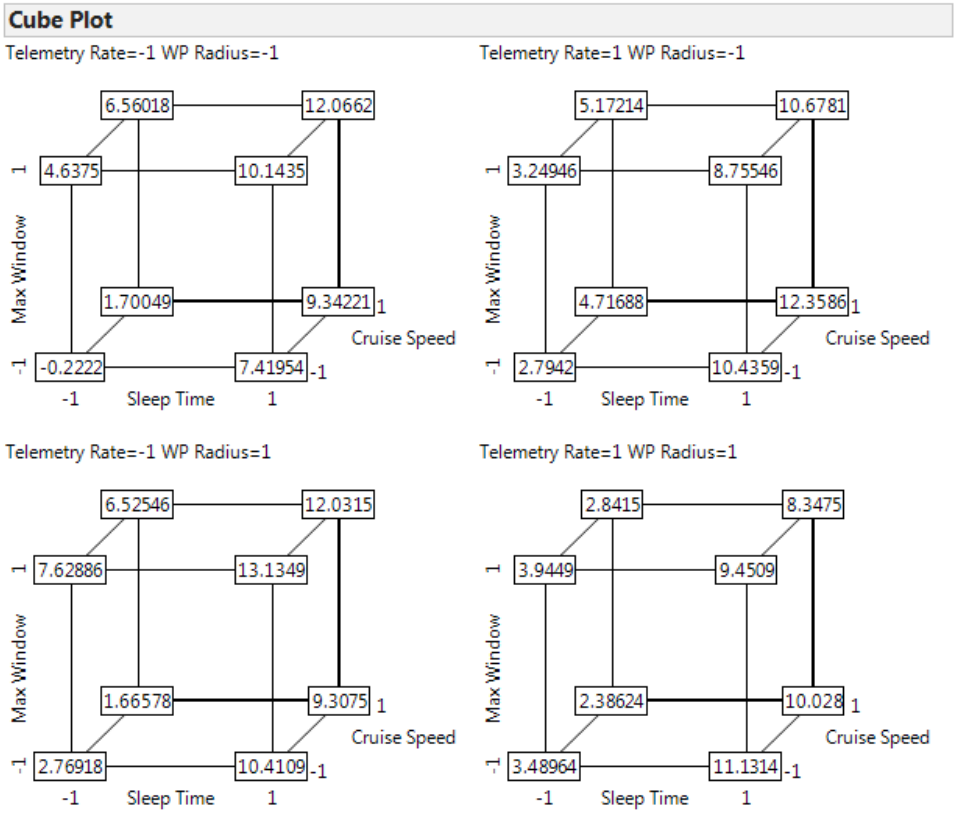

Figure 31. Latency Cube Plot

Notice that in the Cube Plot, when the value of Sleep Time changes from low to high value, there seems to be the largest jump in latency than any other factor change. This validates the significance of Sleep Time. Max Window appears to have the second largest significance, especially seen with low Telemetry Rate. These factors appear to be robust to the remaining three factors in the design. The optimal value of the remaining three factors depend on the settings of other factors, as evaluated by the significant interaction terms.

The interaction plots shown in Figure 32 characterize how the factors interact with each other. Notice that the plots with faded lines involve insignificant interactions in the model. This can also be seen by the parallel lines of latency response as their coded values switch from low to

high values. Therefore, all the plots with line intersections or non-parallel slopes show significant interactions. The more opposite the slopes, or higher change, of the coded factors in the plot, the more significant the interaction. The factor labels on the right side of the graph represent the coded values in the graphs seen laterally. The factor labels seen within squares of the plots represent the interacted factor, and their coded value axis is seen at the very bottom of the plots. For instance, looking at Telemetry Rate's interaction with Max Window, at a low coded value of Telemetry Rate, there appears to be an increase in latency as Max Window changes from lower to higher coded values. Not only is there an increase in latency, but the rate of change, or slope, appears much greater than when Telemetry Rate is at a high coded value. At a high Telemetry Rate coded value, the latency decreases at a slower rate as Max Window changes from low to high coded values. However, the disparity between the high and low coded values of Telemetry Rate and its interaction with Max window appears to be greatest than any other interaction. Therefore, this interaction also appears to be the most significant out of the model which can also be validated from Figure 21's small p-value seen by the interaction.

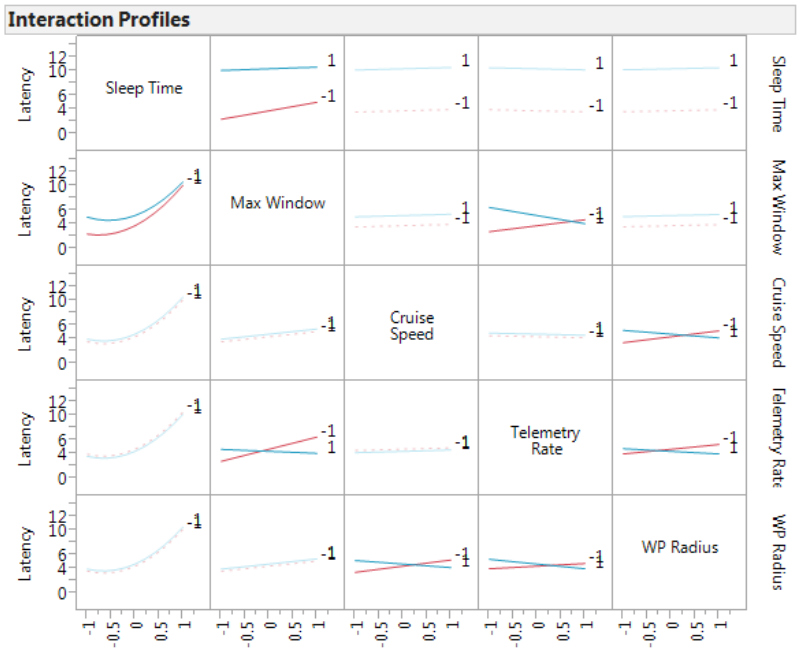

Figure 32. Latency Interaction Plots

The latency model proved to be a very reliable model. With such a high $R^2_{adjusted}$ and no model deficiencies, the model proved to be useful. Yet, a definitive screening design probably should have been used to be able to properly estimate the quadratic terms in the model because there appeared to be curvature. A definitive screening design has three levels and would have provided estimates of the quadratic effects with only partial aliasing. Since one of the reasons a two level design was chosen was in the interest of time, a definitive screening design would have still been a time efficient choice with few runs needed. A two level design was good for sequential experimentation, or follow on experiments. However, no follow on experiments were planned or executed. Even though the knowledge of a definitive screening design was not apparent at the time, the two level latency model still proved to be a very effective model.

The lowest latency was predicted with all factors at negative one coded values except Sleep Time at a -0.763198 coded value. With the design being a two level design, all factors were kept at negative one coded values to achieve optimal latency settings, seen from the cube plot in Figure 31 and Table 8, for the heterogeneous vehicles implementation. Therefore, the heterogeneous vehicles implementations were somewhat used as confirmation runs to validate the predicted latency response for the lowest factor settings, as well as for comparison between vehicle configurations.

Table 8. Optimal Factor Settings for Low Latency

	WP Radius (m)	Cruise Speed (m/sec)	Sleep Time (ms)	Max Window (ms)	Telemetry Rate (Hz)
Coded Value	-1	-1	-1	-1	-1
Actual value	0.25	1	500	33	3

Accuracy Error

Unlike the latency model, the accuracy model didn't include center runs. The omission was made in the interest of time and the lack of trust in the accuracy measuring method. However, there were two extra replicates collected per run, though the three data points for each run were averaged into one response for each run in a 16-run design. The accuracy error model didn't turn out to fit the data as well as the latency model. Seen in Figure 33, the $R^2_{adjusted}$ is 0.379125. This is much lower than the latency model's. The model still proved significant from analyzing the p-value from the Analysis of Variance (ANOVA) table in Figure 34. Only two factors prove to be significant from Figure 35, Waypoint Radius and Telemetry Rate.

Summary of Fit	
RSquare	0.461908
RSquare Adj	0.379125
Root Mean Square Error	24.32612
Mean of Response	55.27599
Observations (or Sum Wgts)	16

Figure 33. Accuracy Error Summary of Fit

Analysis of Variance				
Source	DF	Sum of Squares	Mean Square	F Ratio
Model	2	6603.723	3301.86	5.5797
Error	13	7692.883	591.76	Prob > F
C. Total	15	14296.606		0.0178*

Figure 34. Accuracy Error ANOVA Table

Effect Tests					
Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
WP Radius	1	1	3577.7601	6.0460	0.0287*
Telemetry Rate	1	1	3025.9626	5.1135	0.0415*

Figure 35. Accuracy Error Effect Tests

There proves to be no lack of fit as the test shows its insignificance in Figure 36. Though no center points were executed, taking the only two significant factors in the model led to repeated runs for these two factors in a sixteen run design. These replications are required for

testing lack of fit. The parameter estimates seen in Figure 37 create the regression model in Equation 13.

Lack Of Fit

Source	DF	Sum of Squares	Mean Square	F Ratio
Lack Of Fit	1	1209.9451	1209.95	2.2396
Pure Error	12	6482.9384	540.24	Prob > F
Total Error	13	7692.8835		0.1603
				Max RSq
				0.5465

Figure 36. Accuracy Error Lack of Fit Test

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	55.275994	6.081531	9.09	<.0001*
WP Radius	14.953595	6.081531	2.46	0.0287*
Telemetry Rate	13.752188	6.081531	2.26	0.0415*

Figure 37. Accuracy Error Parameter Estimates

$$x_1 = WP\ Radius, x_2 = Telemetry\ Rate,$$

$$y = Accuracy$$

$$y = 55.275994 + 14.953595x_1 + 13.752188x_2 \quad \textbf{(13)}$$

The regression model shows that to get a desired low accuracy error for the response, low coded values for the factors are desired. The optimal factor settings are shown from the Prediction profiler in Figure 38. Negative one coded values for the two factors gives a predicted minimum accuracy error of 26.57021 inches, or 0.67488 meters. This is also seen from the Cube Plot in Figure 39. The lowest accuracy error appears in the lower left corner of the cube, at the low coded values for both factors. There is a pretty steady increase in accuracy error switching from low coded values to high coded values for each factor, showing the factors' significance to the model.

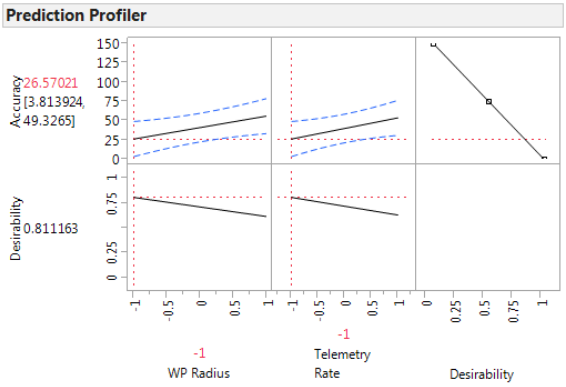

Figure 38. Accuracy Error Prediction Profiler

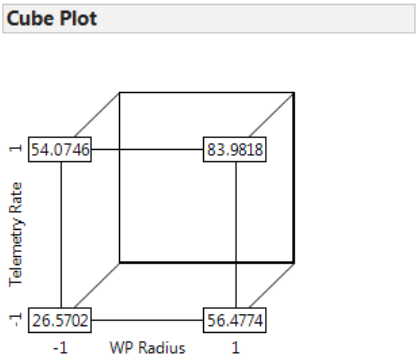

Figure 39. Accuracy Error Cube Plot

The two significant factors, Waypoint Radius and Telemetry Rate, can be rationalized. Having a low Waypoint Radius allows for a more accurate target location. Setting a high Waypoint Radius could keep the rover five meters off of its target because the rover will detect its target location arrival once it reaches the Waypoint Radius of the target. The Telemetry Rate focuses on how often the position data is sent to the GCS [34]. It would be expected that a high Telemetry rate would be better for position accuracy because the position of the vehicle would be updated to the GCS more frequently; however, the model does not show this. In fact, the model suggest a lower Telemetry Rate is more desirable. Perhaps the higher telemetry rate interferes with Mission Planner's ability to service or run Python scripts. If Mission Planner is getting too much information in too little time, this could overload the GCS.

Looking at the Residual vs Predicted plot in Figure 40 and the Normality plot in Figure 41, it can be seen that the Residual vs Predicted plot looks to have a funnel shape with its data points rather than a desired constant variance. This also shows that quadratic terms may have been needed to model this funnel shape. Again, it was too late to run an experiment with three level factors by the time it was realized it may have been needed. The normality plot doesn't have any issues with just a slight data point deviation from the linear line near the middle of the graph. This somewhat unfavorable model adequacy check, coupled with a much lower $R^2_{adjusted}$ than the latency model, raises questions as to whether a better model is obtainable for accuracy error. Therefore, the Box-Cox Transformation seen in Figure 42 shows possible model transformations.

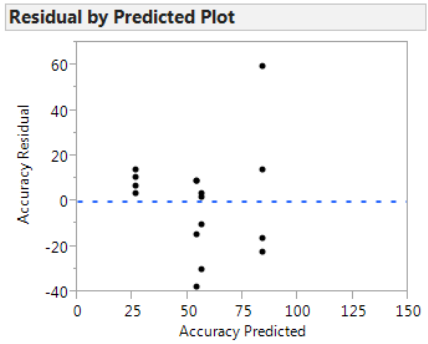

Figure 40. Accuracy Error Residual vs. Predicted Plot

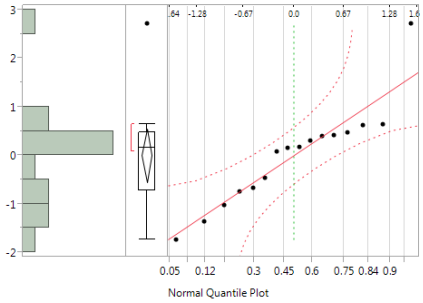

Figure 41. Accuracy Error Normality Plot

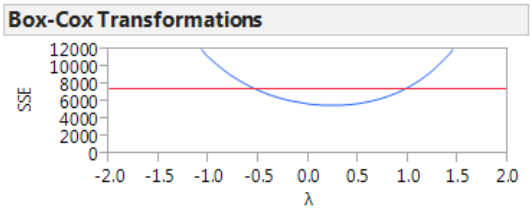

Figure 42. Accuracy Error Box-Cox Transformation

The Box-Cox shows how a transformation on the response of y^{λ} could be made. Figure 42 shows the desired values for λ. The desired λ is the value where the minimum Sum of Squares Error (SSE) value is reached. Usually taking any λ where the SSE is below the red line can be sufficient; therefore, taking 0.5 as λ, a square root transformation on the response is done first. It's seen in Figure 43, that the $R^2_{adjusted}$ hasn't changed much from 0.379125; in fact, it's gotten a bit lower. The Telemetry Rate's significance became a little less as well in Figure 44 by crossing the 0.05 threshold that the p-value usually identifies as significant. The Residual vs Predicted plot in Figure 45 doesn't show much change from the pre-transformed model. Therefore, the square root transformation is deemed unnecessary.

Summary of Fit

RSquare	0.460461
RSquare Adj	0.377455
Root Mean Square Error	1.515997
Mean of Response	7.198266
Observations (or Sum Wgts)	16

Figure 43. Square Root Accuracy Error Transformation Summary of Fit

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	7.1982655	0.378999	18.99	<.0001*
WP Radius	0.9630841	0.378999	2.54	0.0246*
Telemetry Rate	0.8161559	0.378999	2.15	0.0506

Figure 44. Square Root Accuracy Error Transformation Parameter Estimates

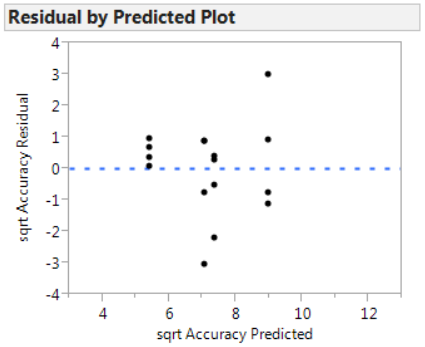

Figure 45. Square Root Accuracy Error Transformation Residual vs. Predicted Plot

Taking a log transformation on the response for y^0 from the Box-Cox graph, the $R^2_{adjusted}$ seems to have dropped quite a bit in Figure 46. The factors have also become less significant in Figure 47. The Residual vs Predicted plot looks to be unchanged in Figure 48. Therefore, this transformation is not desired either. Transformations aim at creating conditions for which the coefficient estimates are accurate and the p-values for significance are valid. From the looks of it, the original model without transformations proves to be the best model to support the data as collected. Introducing quadratic terms may have improved the model though. With the lowest accuracy error desired, the model predicts the lowest accuracy error with the factor settings in Table 9.

Summary of Fit	
RSquare	0.412178
RSquare Adj	0.321744
Root Mean Square Error	0.433122
Mean of Response	3.882944
Observations (or Sum Wgts)	16

Figure 46. Log Accuracy Error Transformation Summary of Fit

Parameter Estimates				
Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	3.8829437	0.10828	35.86	<.0001*
WP Radius	0.2622665	0.10828	2.42	0.0308*
Telemetry Rate	0.1951739	0.10828	1.80	0.0947

Figure 47. Log Accuracy Error Transformation Parameter Estimates

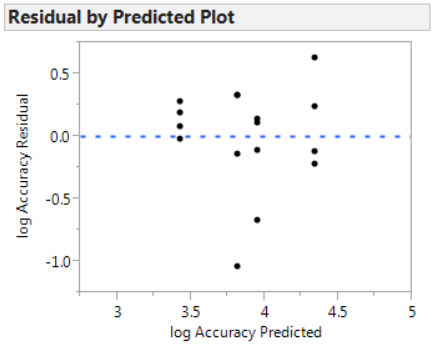

Figure 48. Log Accuracy Error Transformation Residual vs. Predicted Plot

Table 9. Optimal Factor Settings for Low Accuracy Error

	WP Radius (m)	Telemetry Rate
Coded Value	-1	-1
Actual value	0.25	3

This method of measuring accuracy error proved to be more of a measurement of GPS error because the GPS locations were compared to actual distances between the vehicles. The $R^2_{adjusted}$ also proved to be a much poorer fit model than the latency model, with few significant factors. This led to low confidence in the model. The distances measured were also when the vehicles had reached a stationary point. Therefore, this didn't account for the latency that would occur when the follower vehicle would be in pursuit of the leader vehicle. To account for this type of error, a set of Figure eight waypoints were assigned to the leader vehicle. The leader vehicle would then operate in Auto mode, following the waypoints while the follower vehicle would follow the leader using the Python script. Waypoint distance was captured in T-log files. In terms of the follower vehicle in this case, the waypoint is the most recently transmitted position of the leader vehicle. Likewise, the waypoint distance recorded in the T-log represented how far away the follower vehicle was from the last known leader vehicle position at all times, without an offset. Of course there is a bit of lag introduced based on latency, but the waypoint distance still measured how far away from the desired target the follower vehicle was at all times,

labeling this distance as accuracy error. This was to prove, more or less, the vehicles' effectiveness and possible application in a close-formation flight with aerial vehicles.

Figure Eight Accuracy Error

By pulling the waypoint distances from the follower vehicle's T-log and averaging them for each run of factor settings, a response of accuracy error was obtained in meters for each run in the 16-run design. Again, no center runs were executed to check for curvature in the model, in interest of time. A linear model was expected to suffice for the figure eight accuracy error. The data was screened as seen in Figure 49, with the Half Normal Plot seen in Figure 50. Seen here, there are not too many obvious possibilities of significant effects.

Contrasts

Term	Contrast	Lenth t-Ratio	Individual p-Value	Simultaneous p-Value
Cruise Speed	0.788148	1.67	0.1031	0.6864
Telemetry Rate	0.463073	0.98	0.3001	0.9962
Max Window	-0.243448	-0.52	0.6247	1.0000
Sleep Time	0.231298	0.49	0.6403	1.0000
WP Radius	0.097114	0.21	0.8401	1.0000
Cruise Speed*Telemetry Rate	-0.849089	-1.80	0.0852	0.6025
Cruise Speed*Max Window	0.339864	0.72	0.4365	1.0000
Telemetry Rate*Max Window	-0.553961	-1.18	0.2242	0.9605
Cruise Speed*Sleep Time	0.095086	0.20	0.8436	1.0000
Telemetry Rate*Sleep Time	-0.184939	-0.39	0.7068	1.0000
Max Window*Sleep Time	0.739189	1.57	0.1214	0.7555
Cruise Speed*WP Radius	0.625852	1.33	0.1763	0.9016
Telemetry Rate*WP Radius	0.249377	0.53	0.6172	1.0000
Max Window*WP Radius	0.313923	0.67	0.4846	1.0000
Sleep Time*WP Radius	-0.110173	-0.23	0.8163	1.0000

Figure 49. Figure Eight Accuracy Error Model Screening

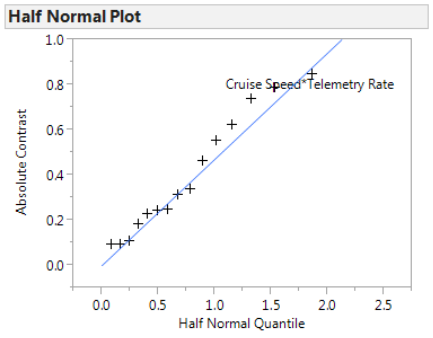

Figure 50. Figure Eight Accuracy Error Half Normal Plot

The model proved to only have one significant main effect in Cruise Speed, as seen in Figure 51. However, all main effects were included in the model for hierarchy due to their factor

interaction significance. Cruise Speed affects how fast the follower vehicle catches up to the leader. Logically, the faster the Cruise Speed, the less waypoint distance is recorded because the follower vehicle will catch up to the leader vehicle in less time. However, the same Cruise Speed was set for the leader vehicle as the follower vehicle to prevent vehicle collision. Therefore, the Cruise Speed relationship or behavior is not as obvious. The model proves to fit the data better than the previous accuracy error method's model with an $R^2_{adjusted}$ of 0.745323 seen in Figure 52. The model has an R^2 of 0.898129. This gap between the $R^2_{adjusted}$ and R^2 shows that too many terms may be in this model. The insignificant main effects included in the model for hierarchy may attribute to this gap. The ANOVA table shows the model's significance in Figure 53.

Effect Tests

Source	Nparm	DF	Sum of Squares	F Ratio	Prob > F
Cruise Speed	1	1	9.938847	11.2388	0.0154*
Telemetry Rate	1	1	3.430992	3.8797	0.0964
Cruise Speed*Telemetry Rate	1	1	11.535236	13.0439	0.0112*
Telemetry Rate*Max Window	1	1	4.909964	5.5521	0.0566
Max Window*Sleep Time	1	1	8.742408	9.8858	0.0200*
Cruise Speed*WP Radius	1	1	6.267043	7.0867	0.0374*
WP Radius	1	1	0.150898	0.1706	0.6939
Max Window	1	1	0.948274	1.0723	0.3403
Sleep Time	1	1	0.855983	0.9679	0.3632

Figure 51. Figure Eight Accuracy Error Effect Tests

Summary of Fit

RSquare	0.898129
RSquare Adj	0.745323
Root Mean Square Error	0.940392
Mean of Response	5.418586
Observations (or Sum Wgts)	16

Figure 52. Figure Eight Accuracy Error Summary of Fit

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Ratio
Model	9	46.779645	5.19774	5.8776
Error	6	5.306018	0.88434	Prob > F
C. Total	15	52.085664		0.0215*

Figure 53. Figure Eight Accuracy Error ANOVA Table

The parameter estimates for the model are seen in Figure 54. The most significant term, which happens to be an interaction term in this case, has the largest absolute value estimate, or coefficient. This helps build the regression model seen in Equation 14.

Parameter Estimates

Term	Estimate	Std Error	t Ratio	Prob>|t|
Intercept	5.4185859	0.235098	23.05	<.0001*
Cruise Speed	0.7881484	0.235098	3.35	0.0154*
Telemetry Rate	0.4630734	0.235098	1.97	0.0964
Cruise Speed*Telemetry Rate	-0.849089	0.235098	-3.61	0.0112*
Telemetry Rate*Max Window	-0.553961	0.235098	-2.36	0.0566
Max Window*Sleep Time	0.7391891	0.235098	3.14	0.0200*
Cruise Speed*WP Radius	0.6258516	0.235098	2.66	0.0374*
WP Radius	0.0971141	0.235098	0.41	0.6939
Max Window	-0.243448	0.235098	-1.04	0.3403
Sleep Time	0.2312984	0.235098	0.98	0.3632

Figure 54. Figure Eight Accuracy Error Parameter Estimates

$$x_1 = WP\ Radius, x_2 = Cruise\ Speed, x_3 = Sleep\ Time, x_4 = Max\ Window, x_5 = Telemetry\ Rate$$

$$y = Figure\ 8\ Loop\ Accuracy$$

$$y = 5.4185859 + 0.0971141x_1 + 0.7881484x_2 + 0.2312984x_3 - 0.243448x_4 + 0.4630734x_5 + 0.6258516x_1x_2 + 0.7391891x_3x_4 - 0.849089x_2x_5 \quad \textbf{(14)}$$

As seen from the regression model in Equation 14 and validated through the Prediction Profiler in Figure 55, as Cruise Speed, Telemetry Rate, and Max Window get smaller, and Sleep Time and WP Radius get larger, a minimum accuracy of 1.971134 m is predicted. The low factor levels are represented by a coded value of negative one.

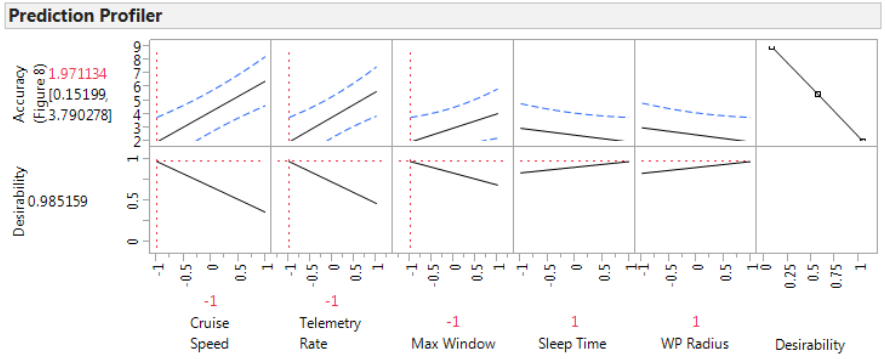

Figure 55. Figure Eight Accuracy Error Prediction Profiler

This minimum accuracy error is also seen from the Cube Plot in Figure 56. Notice how the accuracy error increases more with Cruise Speed than any other factor when the factor is changed from low to high coded values, especially with a high WP Radius. This validates Cruise Speed's significance in the model.

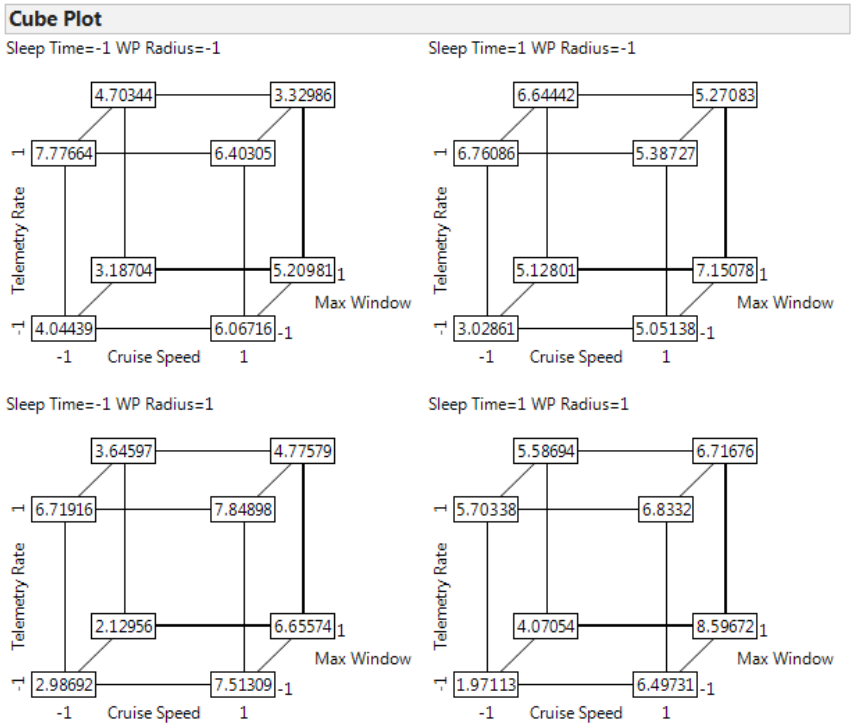

Figure 56. Figure Eight Accuracy Error Cube Plots

The interaction plots shown in Figure 57 show which factor interactions are significant and how significant they are. Cruise Speed*Telemetry Rate, Max Window*Sleep Time, and Cruise Speed*WP Radius appear to be the only significant interactions from Figure 51; yet, the Telemetry Rate*Max Window does appear to have quite a bit of interaction from the interaction plot and is on the borderline of significance.

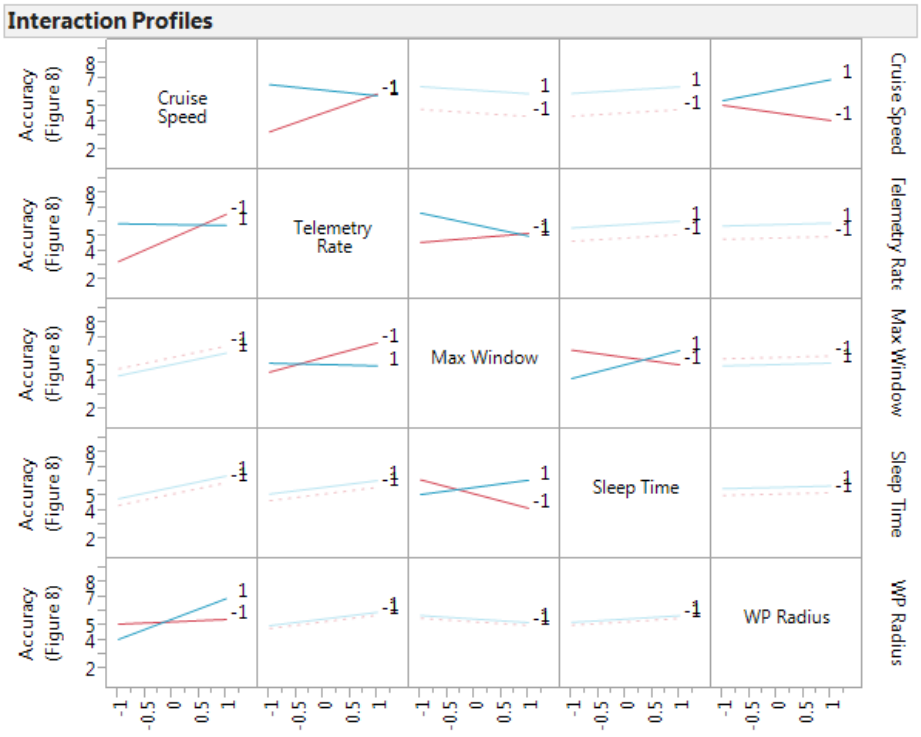

Figure 57. Figure Eight Accuracy Error Interaction Plots

The Residual vs. Predicted plot is seen in Figure 58 and the normality plot in Figure 59 to check the model's adequacy. The Residual vs. Predicted plot looks to have a constant variance, but the normality plot looks to have several points around the center and towards the upper right of the graph that indicate a violation of normality. A transformation is investigated by examining the Box-Cox transformation plot in Figure 60. Since part of the SSE values that are under the red line include $\lambda=1$, there appears no need for a transformation because $y^1=y$.

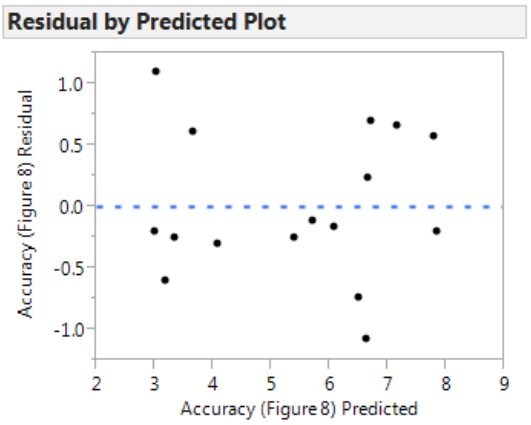

Figure 58. Figure Eight Accuracy Error Residual vs. Predicted Plot

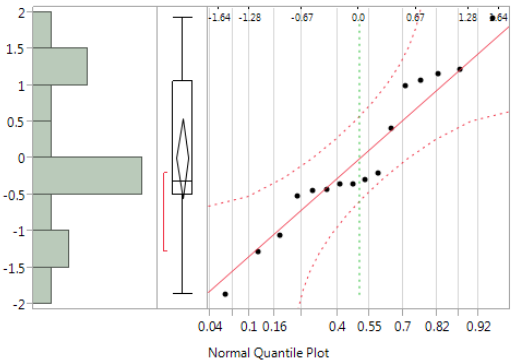

Figure 59. Figure Eight Accuracy Error Normality Plot

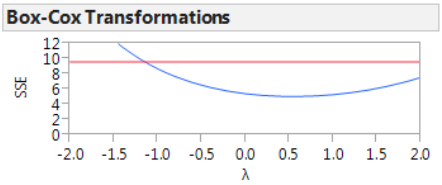

Figure 60. Figure Eight Accuracy Error Box-Cox Transformation

The optimum factor values for low accuracy error for the figure eight method are seen in Table 10. Cruise Speed was discussed earlier as the only significant main effect. Since both the leader and vehicle had the same Cruise Speed at all times, a high Cruise Speed didn't necessarily give an advantage towards a lower accuracy error. This was seen with the figure eight method. In fact, a lower Cruise Speed aided in predicting a lower accuracy error. If both vehicles were traveling at slower speeds, then the follower vehicle didn't have too far to correct its path toward

the leader vehicle if the leader vehicle turned or executed some other, sometimes erratic, maneuver. When both vehicles were set to higher Cruise Speeds, there appeared a much larger distance between the leader and follower vehicles at most times because of the latency involved. The follower vehicle had much more direction and distance to correct for when the leader was traveling at fast speeds. This is an indication that close-formation flight may not be obtainable with the methods investigated here. Cruise Speeds considered lower for aerial vehicles are usually considered faster speeds for ground rover vehicles since some aerial vehicles, such as planes, need to maintain faster speeds for level flight. Aerial vehicles maintaining such low speeds could also be a threat to enemy detection in hostile environments or mission failure due to loss of battery power. The figure eight accuracy error model ended up being used as the primary accuracy error model for heterogeneous vehicle implementation based on the model's improvement over the previous accuracy model.

Table 10. Optimal Factor Settings for Low Figure Eight Accuracy Error

	WP Radius (m)	Cruise Speed (m/sec)	Sleep Time (ms)	Max Window (ms)	Telemetry Rate (Hz)
Coded Value	1	-1	1	-1	-1
Actual value	5	1	5000	33	3

Though the model considered the WP Radius main effect insignificant, there was a visually observable difference when observing the effects of WP Radius on the vehicles, especially the multi-rotor aerial vehicles. When the WP Radius was low, the multi-rotor would exhibit jerking motions while following the ground rover vehicle. When the WP Radius was high, the multi-rotor would follow the ground rover vehicle more smoothly, without jerking motions. This jerking motion could be attributed to the waypoint being changed too much. In other words, the waypoint radius was too precise given the accuracy of the GPS measurement.

The multi-rotor had to constantly re-calculate its target. If a waypoint radius is less than the GPS accuracy, it will constantly change the leader waypoint, even when the leader is stationary. With a high WP Radius, the multi-rotor had time to react to the changing waypoints because the waypoints weren't as precise due to the buffer added from radius length. Though the model displayed an $R^2_{adjusted}$ of 0.74, it was still much higher than the original accuracy model's $R^2_{adjusted}$ of 0.379, proving to be a better fit model. With the lack of a better method for measuring accuracy error, the figure eight accuracy model was used throughout this research.

Heterogeneous Vehicle Implementation

After finding the optimum parameter settings for low latency from Table 8 and accuracy error from Table 10, the settings were used on different vehicle configurations and posed as confirmation runs. Though the models' predicted responses didn't match the collected values, the factor settings did accomplish the objective by producing the desired lowest responses seen from the vehicles tested. Using the Python method, the multi-rotor vehicle was first set as the leader and the ground rover as the follower vehicle. Then the rover ground vehicle was set as the leader and the multi-rotor vehicle as the follower vehicle as seen in Table 11. Finally, the multi-rotor following the rover configuration was used with an alternate method, the Droid Planner 2 application's "Follow-Me" mode seen in Table 12. For the latency tests, three data points were captured for each setting and configuration and then averaged into one data point.

While all optimum parameter settings were used for the tests, a test with Telemetry Rate at its low coded value, or negative one, and a test with its high coded value, or positive one, were captured with each configuration. This was led by curiosity as to whether high or low Telemetry Rate is really desired. As seen from the original accuracy error model that was later replaced, accuracy error favored a low Telemetry Rate. Though this was the only model with a significant Telemetry Rate main effect, the latency and figure eight accuracy error model both favored

having a low Telemetry Rate as well. Logically, one might think a higher Telemetry Rate would reduce latency and accuracy error. Yet, as stated before, perhaps too high of a Telemetry Rate overloads Mission Planner with too much information in such a short time.

Table 11. Heterogeneous Vehicle Implementation

	Latency (sec)			
	Low Telemetry Rate	Std. Dev.	High Telemetry Rate	Std. Dev.
Rover following Multi-Rotor	2.49	0.46	2.61	0.24
Multi-Rotor following Rover	5.16	0.92	4.853333	0.43
	Accuracy Error (m)			
	Low Telemetry Rate	Std. Dev.	High Telemetry Rate	Std. Dev.
Rover following Multi-Rotor	4.89	3.13	4.44	4.36
Multi-Rotor following Rover	2.76	2.5	6.65	3.71

Table 12. Multi-Rotor Following Rover Droid Planner 2 Tests

	Latency (sec)	Std. Dev.	Accuracy Error (m)	Std. Dev.
Droid Planner 2 app	6.75	1.03	0.8	0.94

Seen from Table 11, the latency was a bit lower for the rover following the multi-rotor when a low Telemetry Rate was used versus a high Telemetry Rate and vice versa for the multi-rotor following the rover, proving Telemetry Rate's insignificance. Yet in both vehicle configurations, the higher Telemetry Rate offered more precise data points seen from the standard deviation. In fact, the standard deviation was almost half that of the low Telemetry Rate. Though the results were kind of opposite between both vehicle configurations, the low standard deviation remained the same for a higher Telemetry Rate. Yet, since all previous models favor using a lower Telemetry Rate, the results and differences in latency seen in Table 11 were not enough to justify using a high Telemetry Rate over a low Telemetry Rate. Further experimentation may be

needed to really clarify which, if any, Telemetry Rate is optimal for low latency. Perhaps there isn't necessarily a better option to pick one Telemetry Rate over the other. This could be indicated by the lack of significance for the Telemetry Rate main effect in the latency model from Figure 21.

The accuracy error results were quite the opposite from the latency results. The standard deviation was lower for low Telemetry Rate. The rover following multi-rotor had a slightly higher accuracy error with low Telemetry rate than high Telemetry Rate. However, for multi-rotor following rover, the accuracy error was about forty percent of the high Telemetry Rate's accuracy error. This drastic decrease in accuracy error and lower standard deviation was enough to further validate using a lower Telemetry Rate for a lower accuracy error. Though, as in the case with latency, further experimentation would always help in validating the results.

With the Droid Planner 2 application, Telemetry Rate and Sleep Time were not used since these two factors were related only to Mission Planner and Python specifically. However, still using the other factor settings from the latency and accuracy error models, the results in Table 12 were obtained. It can be seen that the latency was noticeably higher with the Droid Planner 2 application than with Python and Mission Planner. However, the accuracy error was much lower coupled with a lower standard deviation. Therefore, it was fairly obvious that the Droid Planner 2 application introduces more latency, but decreases accuracy error dramatically. This is odd, in the sense, that latency is usually reflected or seen in accuracy error. Perhaps, the latency tests required more processing from the application because the ground rover vehicle was driven out such a far distance at a fast rate away from the multi-rotor. Once the multi-rotor started following the ground rover vehicle, the multi-rotor looked to keep up with the ground rover vehicle very well, possibly because the multi-rotor was already moving in comparison to the latency tests where the multi-rotor is originally stationary. The standard deviations are noticeably larger with accuracy error than latency. Though several of the standard deviations are

close to their accuracy error values, the Droid Planner 2 measurement is the only accuracy error with a higher standard deviation than accuracy error. The standard deviation of Droid Planner 2's accuracy error is still much lower than the other configurations'. These relatively high standard deviations show how varied the waypoint distances are. Still, this leaves concern over the accuracy error measurement methods.

Vehicle Configuration Issues

When executing the heterogeneous vehicle implementation, the ground rover vehicle following the multi-rotor vehicle configuration had no problems with the Python script. However, when executing the multi-rotor following the ground rover vehicle configuration, there were issues. When starting the Python script, the multi-rotor would immediately overrun the safety pilot's manual radio controls and try to land at its set home location. At first thought, the Python script was questioned. After running a similar Python script with only the multi-rotor connected, the multi-rotor worked fine. Mission Planner's Guided Mode was tested out by pointing to a location on the map and sending the multi-rotor to the location by selecting "Fly-to-Here." The multi-rotor, once again, executed properly. Network issues were also investigated between the GCS of the leader and follower vehicles when both were on at the same time. When both vehicles were on and connected to Mission Planner on each of their own GCS at the same time, even the Guided Mode's "Fly-to-Here" command didn't work for the multi-rotor. In fact, the multi-rotor immediately tried to land at its home location. It was later concluded that the vehicles must be connected in a proper sequence, with the multi-rotor being first to connect, in order for the Python script and Guided mode to work effectively. The Droid Planner 2 application also appeared to require a "Fly-to-Here" command to be issued in order to send the multi-rotor into Guided Mode. Once successfully in Guided Mode, the multi-rotor could execute "Follow Me" without the multi-rotor trying to land at its home location again.

Commanded Offset Versus Actual Distance Accuracy Method

Having experienced the results of the two accuracy measurement methods, another accuracy measurement method was applied post-experimentation. The actual distances measured from the three GCS configurations in test one were compared to the commanded relative offsets in Table 13. The absolute value of the differences between these measurements were considered the accuracy error in meters. The relative commanded offset measurements were perceived offset distances between the leader and follower vehicles. Documentation never provided the actual offset units for Python scripting and Mission Planner swarm. The relative error was measured between the vehicles by removing GPS error from the accuracy method. Assuming the same type of GPS receiver on each vehicle, the GPS error of one vehicle was assumed to be similar to the other vehicle in the configuration since both GPS receivers would be communicating with the same GPS satellites. The data from test one was used here because it was the only test where an offset was used. The distance measurements are just magnitudes since the angles or directions were not recorded. As can be seen below, the accuracy error is much less than the original accuracy error measurement method seen from Table 6.

Table 13. Commanded Offset vs. Actual Distance Accuracy

	Observation	Actual Distance (m)	Commanded Offset (m)	Accuracy Error (m)
Python MP on 2 PC	1	1.14	1	0.14
	2	2.08	1	1.08
	3	0.94	1	0.06
	4	1.09	1	0.09
	5	1.4	1	0.4
	Average	1.33	1	**0.36**
	St. Dev	0.45	0	**0.43**
Python 2 MP on 1 PC	1	1.4	1	0.4
	2	1.78	1	0.78
	3	2.03	1	1.03
	4	0.89	1	0.11
	5	1.4	1	0.4
	Average	1.5	1	**0.54**
	St. Dev	0.43	0	**0.36**
MP Swarm	1	2.59	2	0.59
	2	1.68	2	0.32
	3	2.95	2	0.95
	4	3.2	2	1.2
	5	1.73	2	0.27
	Average	2.43	2	**0.67**
	St. Dev	0.7	0	**0.4**

By finding the desired mean point from the commanded offset. The range error probable (REP), deflection error probable (DEP), and circular error probable (CEP) can be found [35]. The range is considered the y axis of a location based on the trajectory of the vehicle, while the deflection is considered the x axis of a location. The REP is the range distance to parallel lines that include 50% of the location points from the commanded offset. The DEP is the same as REP, but is deflection distance versus range distance. The CEP is the radius of a circle that includes 50% of the location points from the commanded offset and is calculated from Equation 15. The relation of CEP to REP and DEP can be seen from Equation 16. These calculations result from REP or DEP containing 50% of the locations or $F(Z) = 0.75$, which is $Z = 0.6745$, from a zero mean normal distribution. These measures give ideas of the dispersion of follower vehicle locations and are seen in Table 14. The larger the CEP values, the more error can be

attributed to the vehicle's recognition of its position. This accuracy method, like the original accuracy method from test one, is a stationary test. Therefore, latency is not factored into the measurements. With latency, accuracy error could increase by multiplying the latency by the velocity of the vehicles.

$$CEP = 1.1774\sigma \tag{15}$$

$$REP = DEP = 0.573 \times CEP \tag{16}$$

Table 14. Commanded Offset vs. Actual Distance Accuracy CEP

	CEP (m)	REP/DEP (m)
Python MP on 2 PC	0.5	0.29
Python 2 MP on 1 PC	0.43	0.24
MP Swarm	0.47	0.27

Summary

Through diagnostic testing, a baseline was established with Mission Planner's Swarm application. Python was used to improve the capabilities of Mission Planner's Swarm application. The Python script written was first tested on two ground rover vehicles. Though the latency was a bit higher than Mission Planner's Swarm application, Design of Experiments was used in order to find the optimal parameter settings in order to lower the latency and accuracy error as much as possible. Sleep Time, written in the Python script, turned out to have the most control over latency. Accuracy error was measured two different ways once the first model proved undesirable and the measuring method unpractical. While the original accuracy error measurements took the GPS location of the vehicles and compared their distance to the actual measured distance, the figure eight accuracy error was measured by collecting the average of the waypoint distances from the follower vehicle's T-log. The figure eight accuracy error model proved much more reliable and practical. Therefore it was used as the primary method for measuring accuracy error in subsequent heterogeneous vehicle implementation tests. The rover

following the multi-rotor vehicle configuration proved to have much lower latency than when the multi-rotor was following the rover. However, there appeared lower accuracy error for the multi-rotor following the rover, at least with low Telemetry Rate. High and low Telemetry Rates were run for each vehicle configuration in order test the logic behind the setting, with the other factor settings remaining the same as the optimum factor settings from produced models. Evidence proved that a low Telemetry Rate was best suited for a low accuracy error. Yet, not enough evidence could justify picking a high Telemetry Rate over a low Telemetry Rate recommended by the latency model. Using the Droid Planner 2 application from a smart phone, another method was introduced to test latency and accuracy error with cooperative behavior and control for heterogeneous vehicles. The application induced more latency between the vehicles. However, the accuracy error dropped significantly. Therefore, this benefits the accuracy error test more than the latency test. After solving several vehicle configuration issues posed earlier, it was concluded that when having multiple vehicles connected to GCSs, Guided Mode requires that a sequence of vehicle connections be made, starting with the multi-rotor aerial vehicle first.

Two accuracy models were used in this research. Incorporating a third accuracy method which compared the commanded offset to the actual distance measurement between the vehicles conveyed the relative error of the vehicles. The accuracy errors were noticeably smaller because the method didn't account for GPS error. GPS error for each vehicle is assumed to have miniscule differences when operating in such close proximity to one another due to the GPS receivers onboard the vehicles viewing the same GPS satellites. Further research could investigate the individual GPS error of each vehicle in a configuration.

The latency experienced from results was reduced using optimal factor settings, but is still too high for certain applications. In the proceeding chapter, the results will be used to analyze applications that may suitable for this research.

V. Application Analysis

Chapter Overview

Once all data had been captured and all experiments run, potential applications for the research can be analyzed. This chapter focuses on the military application of this research and the analysis involved with its use. Three different applications are discussed and analyzed based on the research gathered. The selected applications are close-formation flight, sharing target information, and vehicle following. The research may or may not have provided evidence to support these applications.

Close-Formation Flight

With the use of cooperative behavior and control, close-formation flight is often considered as a possible application. Precision flight requires very low latency for immediate response times. When in close formation, if any vehicle in formation exhibits higher levels of latency, the whole formation could be in danger of collisions. Close formation flights usually require a minimum response rate of 10 Hz to an optimum rate of 60 Hz for effective use [36]. Unfortunately, no rate above 0.5 Hz was attainable from the experiments. This two second minimum latency is much too slow and dangerous for any kind of close formation flight. If there are other ways of reducing the latency, perhaps using direct vehicle to vehicle communication and on-board processing, then there may be a possibility to support close formation flight with low cost vehicles in the future. Mavlink cuts out the middle man, in this case the Mission Planner, between the GCS and the autopilot. Therefore, Mavlink can send messages and commands directly to the autopilot instead of having to navigate through Mission Planner, which can cause higher latency due to processing requirements and GUIs associated with the software [37].

Target Information Sharing

Though the latency observed with this research may be an issue with close formation flight, there may be other applications out there where low latency may not be a necessity. Target information sharing from the vehicle to the GCS, and between vehicles, is vital in military operations. When the vehicle detects a target, it should be able to transmit the general location of the target to the GCS or between vehicles for a cooperative search. In this scenario, the amount of latency shown through this research shouldn't be an issue if the vehicle is just transmitting target position.

Vehicle Following

One of the primary applications demonstrated by this research is vehicle following. Whether it be a vehicle following a friendly vehicle, or a vehicle following some other target, vehicle following could be militarily useful. The use of heterogeneous vehicles, aerial and ground vehicles in this case, give two different perspectives of an area or target. Though an aerial target may cover more area, the ground target could offer closer and clearer views of Points of Interest (POI), depending on the cameras used. The use of heterogeneous vehicles could allow some vehicles to travel in terrain where others cannot. With cameras installed on a wide variety of drones in operation, it is obvious that surveillance is a popular application not only in the civilian world, but in the military realm as well.

Given the rover ground vehicle speeds and latency received from testing and experimentation, camera angles can be adjusted for on-board cameras on aerial vehicles to produce different footprint projections. By calculating these footprint sizes, the time it takes for a rover to exit the footprint can be calculated, assuming a stationary multi-rotor and camera. This time can be compared to experienced latency between the vehicles in order to find the time the multi-rotor would have to respond to the rover before the rover exited the footprint. If the rover

were ever to exit the footprint, the time it would take for the multi-rotor to get its camera within FOV of the rover could lead to a loss of visibility during a critical mission time or a target vehicle eluding surveillance.

To analyze the vehicle following application, a pixel density of 40 pixels/m^2 minimum on target will be used for the application because this is approximately the density required to identify and distinguish between vehicle targets [38]. The required pixel density results in a Ground Separation Distance (GSD) of 0.16 m/pixel to use in other calculations, as seen from Equation 17 [38].

$$y = \frac{40\ pixels}{m^2} \rightarrow \left(\frac{6.32\ pixels}{m}\right)^{-1} = \frac{0.16m}{pixel} = Required\ GSD \quad \textbf{(17)}$$

The pixel arrays are assumed to be standard high definition (HD) arrays having 1024 lines of horizontal resolution and 768 lines of vertical resolution. The azimuth Field of View (FOV) will be chosen as either thirty degrees, or sixty degrees. The pixel spacing, in pixels/degree, for each azimuth angle are calculated by dividing the azimuth pixel count by the azimuth FOV, in degrees. This same pixel spacing will be assumed for elevation, with a proportional reduction in elevation FOV based on 768 lines of resolution (versus 1024 lines in azimuth). By dividing the elevation pixel length, 768, by the pixel spacing, in pixels/deg, the elevation FOV, in degrees, will be found. The angular spacing between each pixel can be found by converting the inverse of pixel spacing, in pixels/deg, to radians, giving radians/pixel. The maximum range, in meters, can then be found by dividing the required pixel density, in m/pixel, by the angular spacing, in radians, seen from Equation 18 [38].

$$R_{max} = \frac{Required\ \ GSD\ (\frac{m}{pixel})}{Angular\ Spacing\ (radians/pixel)} \quad \textbf{(18)}$$

A visual of the multi-rotor and its camera calculations can be seen in Figure 61. The following results of the calculations between the two azimuth FOVs can be seen in Table 15. As the azimuth FOV decreases, the pixel spacing and maximum range increase.

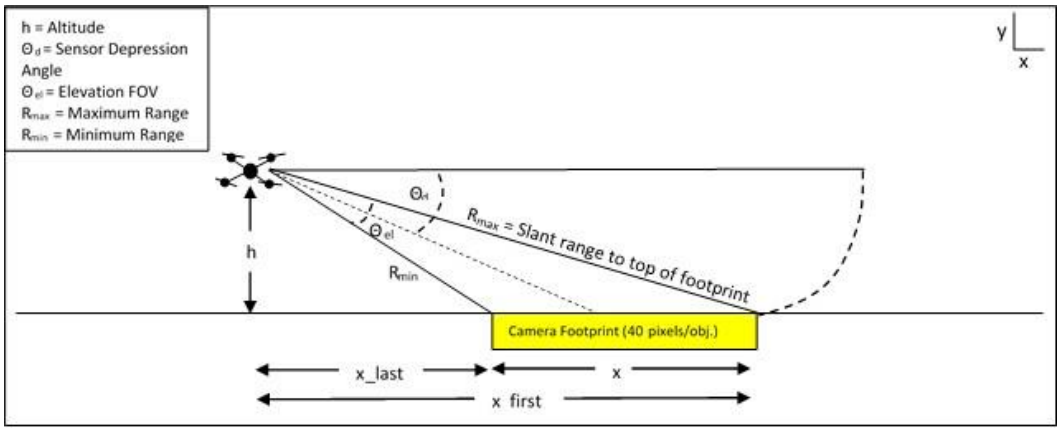

Figure 61. Sideview of Camera Footprint

Table 15. Angle Calculations for Azimuth FOV

	Azimuth FOV (degrees)	
	30	60
Pixel spacing (pixels/deg)	34.13	17.07
Elevation FOV (degrees)	22.5	45
angular spacing (radians/pixel)	0.00051	0.00102
Maximum Range (m)	309.22	154.61

Two altitudes, *h*, are chosen for calculation, a high, 100 m, and a low, 50 m. For each azimuth FOV-altitude configuration, a sensor depression angle, in degrees, a minimum range, in meters, an *x_last*, in meters, *x_first*, in meters, and *x*, in meters was calculated. The ground distance from the camera to the start leading edge of the footprint on the ground, in meters, is found in *x_first*. The ground distance from the camera to the end trailing edge of the footprint on the ground, in meters, is found in *x_last*. The sensor depression angle, in degrees, was found using Equation 19 [38]. This will set the maximum range to the same length as the slant range to the top of the scanned footprint. This will allow for the rover ground vehicle to be within maximum range at all footprint locations to support the required level of target discrimination.

$$Sensor\ Depression\ Angle = \frac{180}{\pi} * sin^{-1}\left(\frac{h}{R_{max}}\right) + \frac{Elevation\ FOV}{2} \quad \textbf{(19)}$$

The minimum range, in meters, is the slant range to the bottom of the scanned footprint. This is calculated using Equation 20 [38]. Using the Pythagorean Theorem, *x_last* and *x_first* are found with previously calculated values, seen in Equation 21 and Equation 22 [38]. The ground distance depth of the footprint, in meters, is measured in *x*, seen from Equation 23 [38]. The calculations from 30 and 60 degree azimuth FOVs, and 100 meter and 50 meter altitude configurations are seen in Table 16.

$$R_{min} = \frac{h}{sin\left(\frac{\pi}{180} * \left(Sensor\ Depression\ Angle + \frac{Elevation\ FOV}{2}\right)\right)} \quad \textbf{(20)}$$

$$x_{last} = \sqrt{R_{min}^2 - h^2} \quad \textbf{(21)}$$

$$x_{first} = \sqrt{R_{max}^2 - h^2} \quad \textbf{(22)}$$

$$x = x_{first} - x_{last} \quad \textbf{(23)}$$

Table 16. Footprint Distances

Azimuth FOV (deg)	Altitude (m)	Sensor depression angle (deg)	R_{min} (m)	x_last (m)	x_first (m)	x (m)
30	100	30.12	151.31	113.55	292.61	179.1
30	50	20.56	94.87	80.62	305.15	224.53
60	100	62.8	100.34	8.22	117.92	109.7
60	50	41.37	55.69	24.53	146.3	121.77

Notice how the footprint grows larger with lower azimuth FOV angles permissible at lower altitudes. The minimum range and x_last also increase as the azimuth FOV decreases and altitude increases. The value for x_first increases when azimuth FOV and altitude decrease. The sensor depression angle decreases as the azimuth FOV and altitude decrease.

Along with the azimuth FOV and altitude, the rover ground vehicle speed and latency can be input to the calculations for analysis. The rover ground vehicle speed, in meters per second, will be altered from 6 m/s, the high speed run from the experiments, to 1 m/s, the low speed run from the experiments. The latency will vary between 2.5 seconds, the lowest latency captured in the heterogeneous vehicle configurations, and 6.75 seconds, the highest latency captured in the heterogeneous vehicle configurations. The track lag, or latency, is a parameter that is observed rather than set. Though Chapter IV discussed optimal settings and configurations for achieved latencies, these latencies cannot be set directly. The footprint of the camera is represented as a trapezoidal shape onto the ground, as seen in Figure 62. Using simple trigonometry, the distances of the edges of these footprints can be solved. Assuming the rover ground vehicles to start at the center of the multi-rotor's camera footprint onto the ground, and the multi-rotor to be stationary, the times until the rover ground vehicle exits the footprint, traveling forward, are seen in Table 17. The times are divided by the track lag, or latency, to create a ratio, or buffer. This buffer shows a ratio of how many times more the time it take for the rover to leave the multi-rotor camera's footprint is than the latency. Alternatively, the difference between latency and the time it takes for the rover to leave the footprint could be used to measure the tolerance for additional latency.

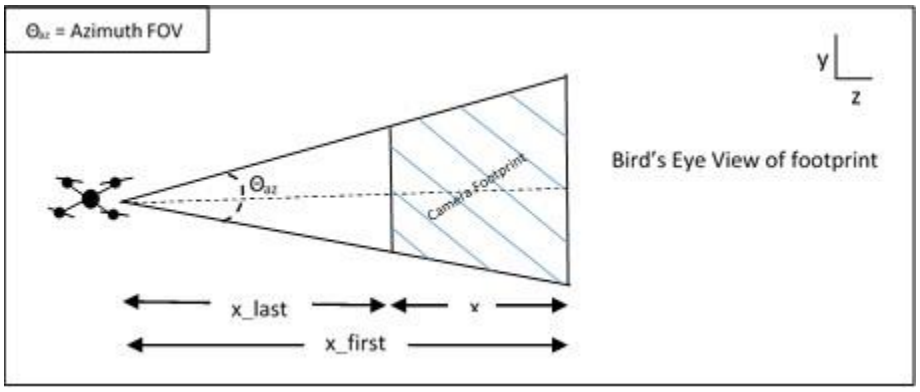

Figure 62. Bird's Eye View of Camera Footprint

Notice how 90° turn times and buffers are calculated too. These calculations, again, assume the rover ground vehicle to start at the center of the multi-rotor camera's footprint and the multi-rotor to remain stationary. However, the rover ground vehicle is now projected to take a 90° turn left or right, instead of traveling forward, until it reaches outside of the multi-rotor camera's footprint.

Table 17. Rover Travel Time/Latency Buffers

Azimuth FOV (deg)	Altitude (m)	Ground Vehicle Speed (m/sec)	Time till out of footprint (sec)	Track Lag (sec)	Buffer	90 degree turn distance (m)	Time (sec)	90 degree turn Buffer
30	100	6	14.92	2.5	5.97	54.42	9.07	3.63
30	100	6	14.92	6.75	2.21	54.42	9.07	1.34
30	100	1	89.53	2.5	35.81	54.42	54.42	21.77
30	100	1	89.53	6.75	13.26	54.42	54.42	8.06
30	50	6	18.71	2.5	7.48	51.68	8.61	3.45
30	50	6	18.71	6.75	2.77	51.68	8.61	1.28
30	50	1	112.26	2.5	44.91	51.68	51.68	20.67
30	50	1	112.26	6.75	16.63	51.68	51.68	7.66
60	100	6	9.14	2.5	3.66	36.41	6.07	2.43
60	100	6	9.14	6.75	1.35	36.41	6.07	0.9
60	100	1	54.85	2.5	21.94	36.41	36.41	14.57
60	100	1	54.85	6.75	8.13	36.41	36.41	5.39
60	50	6	10.15	2.5	4.06	49.32	8.22	3.29
60	50	6	10.15	6.75	1.50	49.32	8.22	1.22
60	50	1	60.89	2.5	24.35	49.32	49.32	19.73
60	50	1	60.89	6.75	9.02	49.32	49.32	7.31

A higher buffer is generally desirable with the calculations. On that note, it is interesting to see that the highest buffer, with the rover ground vehicle traveling forward, is 44.91. This is with an azimuth FOV of 30 degrees, an altitude of 50 meters, a speed of 1 m/sec, and a latency of 2.5 seconds. The buffer for the associated 90° turn is 20.67. Though high, this is still not the highest buffer for the 90° turns. The highest buffer under the 90° turns happens to be 21.77 with an azimuth FOV of 30 degrees, an altitude of 100 m, rover ground speed of 1 m/sec, and latency of 2.5 seconds. The associated buffer for the rover ground vehicle traveling forward, with the

same settings, is 35.81. Therefore, the data show that at the highest 90° turn buffer, the high 90° turn buffer is associated with the high altitude. As the multi-rotor flies higher, the horizontal edges of the footprint extend. However, it is seen that the vertical edges of the sufficient resolution footprint do not extend as fast as the horizontal edges with increased altitude because the associated rover ground vehicle traveling forward has a buffer that is not the highest of the forward buffers collected. This is further demonstrated with the highest forward traveling buffer, which is at a 50 meter altitude. The associated 90° turn buffer for the same settings happened to be the second highest buffer as well. The lower azimuth FOV angles appear to offer higher buffers because the footprints are larger.

It is also obvious to see that the highest buffers are generally associated with low rover ground vehicle speed and low latency. The highest buffers for a rover ground vehicle traveling at a speed of 6 m/sec are 7.48 for the forward traveling vehicle and 3.63 for the 90° turn buffer. These are much lower buffers than received with lower cruise speeds. The highest buffers for a rover ground vehicle traveling with a 6.75 second latency are 16.63 for the forward traveling vehicle and 8.06 for the 90° turn buffer. These are still quite a bit lower buffers than received with lower latency.

Investigative Questions

This research set out to answer a list of investigative questions in order to respond to the research's primary question. What methods are currently used for cooperative behavior and control with low cost vehicles? Seen primarily from chapter 2, there were several methods researched that involve cooperative behavior and control. One of these methods was actually used in this research. Mission Planner's swarm application offers two vehicles to connect to a single instance of Mission Planner. This application is essentially a follow-the-leader application

that can assign an offset to be maintained between the leader and follower vehicle at all times. Wherever the leader vehicle goes, the follower vehicle follows in a geodetic frame. Therefore, the offset only allows the follower vehicle to follow the leader in terms of a North, South, East, or West offset. For example, if the leader were to turn from a North to East direction, the follower vehicle would maintain its offset from the leader, but would turn East with the leader, maintaining its same coordinate offset.

What are the challenges of using multiple heterogeneous vehicles from a single GCS? Research shows that military flight restrictions limit the use of multiple aerial vehicles to one aerial vehicle per GCS or operator [5]. This was the underlying restriction heading into this research. However, it was found by using two rover ground vehicles on a single GCS, that latency was actually increased 50%. This was found using the Python script method. The theory lies in an increased processing requirement when two instances of Mission Planner are run from the same GCS, therefore inducing lag. Surprisingly, perhaps not having a wireless network connection between two GCS may inhibit the effectiveness of cooperative behavior and control. Though the research only experimented with two vehicles connected to a GCS at the same time, an increase in vehicle connection could further limit vehicle response time as well as operator response time.

What is the initial architecture that can be implemented and improved upon? This was found from Mission Planner's Swarm application. The architecture was seen from Figure 10 in chapter three. This baseline architecture was used to create a new Python method, developed by programming similar behaviors of the swarm application and improving upon it. Some of these improvements included allowing a heading offset, instead of a geodetic frame offset, between the leader and follower vehicles, and introducing a safety switch into the script to allow the ability for the safety pilot to regain manual control in precarious situations. These were a couple of

behaviors Mission Planner's Swarm application didn't have, making it a risky method, but leaving potential for further improvement.

What appropriate assessment measures should be used for analysis? Latency and accuracy error were used in this research to measure the effectiveness of cooperative behavior and control. Research demonstrated that low latency was a requirement in several cooperative behavior and control applications. For instance, close-formation flying required a 10-60 Hz response time for effectiveness. Low accuracy error is also warranted for precision based scenarios, also demonstrated from close-formation flying. The aerial vehicles must maintain a very precise and accurate offset between other vehicles in formation to prevent collision. When vehicles are in a following configuration, accuracy error is necessary, especially when an offset must be maintained. However, it was seen from chapter 2, that perhaps human effectiveness should also be monitored unless a fully autonomous configuration is to be used.

What are the performance limitations given current architecture? The architecture used for DOE and heterogeneous vehicle configurations was seen in Figure 12 from chapter three. Latency seemed to be an issue between vehicle configurations. Some latencies experienced during experimentation were as high 10+ seconds. However, the latency was able to be brought down between 2-3 seconds, which is still high, especially for certain applications like close-formation flight. Accuracy error was brought down to below five meters, which is about the standard error for GPS.

With heterogeneous vehicle configurations, a specific sequence of vehicle connections was required in order for Guided Mode to work effectively. Whenever an aerial multi-rotor was used, it always had to be connected first if used in conjunction with rover ground vehicles. The Python method would override the safety pilot's manual controls. Therefore, as stated earlier, a safety switch had to be programmed into the script.

The lowest latency values were achieved with low cruise speeds, which could be ineffective for fast-paced military applications. However, cruise speed didn't seem to have as large of an effect towards latency as other factors. The lowest accuracy errors were achieved with a high sleep time. This is directly in confliction with the low latency settings. However, again, sleep time seems to not have a very large effect on accuracy error. The low accuracy error model exhibited a desired high waypoint radius for the follower vehicle; it was seen that a low waypoint radius would give a repeated jerking motion to the vehicle as it followed the leader. This was due to the waypoint radius being so small, that it was continually trying to find a stable point. In this instance, the waypoint radius was probably much smaller than the average GPS accuracy.

What cooperative behavior applications are reasonable or achievable given current limitations? As seen from chapter 2, and earlier in this chapter as well, close-formation flight was a possibility heading into this research. However, it was concluded that the 0.5 Hz rate shown from the heterogeneous vehicle configurations is much too low for the 10-60 Hz close-formation flight requirement. Yet, target information sharing proved to be a potential application because a general target position would be all that is needed to be shared with the GCS or other vehicles. The latency exhibited from the research should not adversely affect the outcome. Still, the most well fit application seems to be with vehicle following. With a camera installed on an aerial multi-rotor, calculations were run through a trade study of the most effective variable settings for the multi-rotor to follow a rover ground vehicle while keeping the rover within the stationary camera footprint. It was seen that with a low azimuth FOV, low altitude, low ground rover speed, and low latency, the highest ratio, or buffer, between time it would take for the rover to exit a stationary multi-rotor camera footprint, and latency would be obtained. Ultimately, this type of vehicle following seems to benefit surveillance missions. Heterogeneous vehicle configurations would allow for differing views of POI. Obviously aerial vehicles would give a bird's eye view,

perhaps covering more area, while a land vehicle would be able to investigate these POI with the ability to get closer to a target getting clearer visuals.

Given the state of technology for commercially available autopilots and Remote Control (RC) hobbyist equipment, what is the achievable performance for cooperative behavior among heterogeneous vehicles? The research was performed to investigate this research question. By successfully answering the investigative questions of the research, the performance of cooperative behavior and control amongst heterogeneous vehicles can be predicted. Lowering the latency and accuracy error provides potential for further improvements, opening the, once closed doors, of applications discussed in the research. This sets the ground for integrating more vehicles into heterogeneous vehicle configurations, as well as integrating new vehicles, such as planes, into these configurations.

VI. Conclusions and Recommendations

Chapter Overview

The chapter discusses the conclusions of the research efforts. The differences between theoretical and recorded data are discussed as well as confidence levels in the recorded data. The significance of the research, such as unexpected results, is communicated. Recommendations for further action are explained, such as how the research could have been performed differently and experiments designed to take the research further. Finally, recommendations for future research are offered.

Conclusions of Research

Given the state of technology for commercially available autopilot and Remote Control (RC) hobbyist equipment, the achievable performance for cooperative behavior among heterogeneous vehicles was observed from the answering of several investigative questions.

Several methods currently used for cooperative behavior and control with multiple low cost vehicles involve Mission Planner's Swarm application and the interaction of several ground vehicles and a multi-rotor to navigate over terrain obstacles. Mission Planner's Swarm application was a baseline architecture that was improved upon using Python programming skills and implemented in the research. Some challenges of using multiple heterogeneous vehicles from a single Ground Control Station (GCS) were found from restrictions written in policy limiting the operation and connection of one aerial vehicle to every GCS, or operator, and an increase in latency response between the leader and follower vehicles. Cooperative behavior and control measures included latency and accuracy error due to their importance in several cooperative behavior and control applications, such as close-formation flight. Given these assessment measures, the performance limitations of the Python method included a rather high latency, the override of the safety pilot's manual radio control of the vehicles, and the

requirement of sequenced vehicle connection in a multiple vehicle configuration. With current limitations, cooperative behavior and control applications such as target information sharing between vehicles and GCSs, and vehicle following were found appropriate from the research.

Significance of Research

The research cannot necessarily be defined as a success or failure based on results, but on accomplishing the focus of the research. The performance of cooperative behavior among heterogeneous vehicles was measured from latency and accuracy error data. Results seemed to defy logic in several circumstances, such as a higher latency with two vehicles connected to a single GCS rather than with each vehicle connected to its own GCS. Using the Python method, a latency increase of 50% was experienced with two vehicles connected to a single GCS rather than with each vehicle connected to its own GCS. The reasoning behind an expected lower latency with the two vehicles connected to a single GCS involved signal reception delays that are characteristic of wireless networks. These wireless networks were used with each vehicle connected to its own GCS. However, the reasoning behind a higher observed latency with the two vehicles connected to a single GCS involves a higher Central Processing Unit (CPU) processing requirement when two instances of Mission Planner and sets of telemetry modems are operating from the same GCS. Perhaps the CPU processing capabilities were not advanced enough to offer lower latencies than with two GCS configuration.

The original accuracy error model contained an unexpected low $R^2_{adjusted}$, meaning the model did not fit the data well. This accuracy error measurement method included recording the GPS coordinates of the two vehicles, leader and follower, calculating the distance, and subtracting from the actual physical distance between the two vehicles. The reasoning behind the poor results of the model was that the measurement method was flawed and perhaps only measured GPS error. This GPS error likely introduced noise into the model.

Another instance of questionable results was from the recommended high waypoint radius and sleep time settings for low accuracy error for the figure eight measurement method. It was originally thought that a more precise target, with a low waypoint radius, and low latency settings, with a low sleep time, would result in lower accuracy error. However, the figure eight accuracy error model favored a high waypoint radius. In the multi-rotor following rover vehicle configuration, the multi-rotor seemed to occasionally experience a jerking reaction when set at a low waypoint radius. This jerking reaction could be attributed to the precision of the waypoint. GPS error is at least a few meters which would create an unstable waypoint in terms of GPS coordinates. The measurement method favored a high sleep time because it allowed more time for the follower vehicle to catch up to its waypoint, before the waypoint was updated again based on the leader vehicle's location. Since the leader vehicle was following a figure eight pattern, by the time the follower vehicle's waypoint was updated, the leader vehicle could have been in a closer location to the follower vehicle.

Original thoughts behind position telemetry rate were flawed as well. Telemetry rate ranges from one hertz to ten hertz. However, the low values of telemetry rate used in the research were three hertz because it is the default. The predicted model for both latency and accuracy error favors a low telemetry rate. This prediction defied the logic that a higher telemetry rate would increase the speed at which information is passed between the GCS and vehicle, thereby lowering latency and accuracy error. Therefore, the telemetry rate was specifically tested, keeping the other factors constant, with the heterogeneous vehicle configurations. The results did not clearly show a lower latency with the either high or low telemetry rates, which could be validated with the factor's insignificance from the model. In fact, the rover following the multi-rotor vehicle configuration had slightly lower latency with a low telemetry rate while the multi-rotor following rover vehicle configuration had slightly higher latency with a low telemetry rate. However, for the multi-rotor following rover vehicle configuration, a low telemetry rate reduced

the accuracy error by more than half of the high telemetry rate's accuracy error. Again, the telemetry main effect was insignificant in the accuracy error model, but was included for hierarchy. An interaction including telemetry rate appeared significant to the model. Therefore, theory concludes that too high of a telemetry rate could possibly lead to instability.

One of the oddest results appeared from the Droid Planner 2 application. The application was used for the multi-rotor following rover configuration. This vehicle configuration did appear to exhibit more latency than with the rover following the multi-rotor. However, the Droid Planner 2 application appeared to create more than a second and a half of extra latency between the vehicles. Yet, this application offered an extremely low accuracy error of less than a meter. Therefore, the Droid Planner 2 application resulted in the highest latency amongst the heterogeneous vehicle configurations, and the lowest accuracy error. Since the accuracy error tests usually factor in latency, the logic seems flawed. Nevertheless, the multi-rotor did seem to follow the rover quite well with the Droid Planner 2 application. Theory behind the low accuracy error results involves using different platforms for GPS measurements. Droid Planner 2 used the smart phone's GPS to identify the GCS target with the Follow Me function, while Mission Planner used GPS measurements from the 3DRobotics GPS connected to the Pixhawk autopilot on the leader vehicle.

The confidence of the data and results still remains high even with unexpected outcomes. Conclusions were drawn above that could explain the results of each action. Three to five replications were performed for each experiment, averaging into one value, to increase confidence. Yet, the results of the Droid Planner 2 application still seem somewhat questionable because the application is fairly new and there was not much experience with the application. The application was used primarily for comparative purposes.

Recommendations for Action

Appropriate sample sizes to detect a specified distance between latency results from test one should have been calculated using power analysis. Instead, an arbitrary five samples were recorded for each configuration. Power is the probability that the test will properly identify a significant difference between configurations, given that a difference actually exists. Large sample sizes usually produce high power, which is desired. Having an appropriate sample size prevents the risk of random data results.

Though the 2^{5-1} fractional factorial design used for Design of Experiments (DOE) resulted in useable models, more suitable designs exist. The fractional factorial design was used to limit experiment runs, lessening experiment execution time. However, the two level design is used with anticipation of sequential experimentation and could not test for curvature or lack-of-fit without a third level from the factors. No sequential experimentation was performed with the models. In the latency model, four center points were used to test for curvature because the latency runs were the quickest to execute and because sleep time's exponential effect on latency looked to possibly cause curvature in the model. Once the latency model detected curvature, there was no way to estimate quadratic terms in the model because the design was a two level design. A minimum of three levels are required in a design to estimate quadratic effects. Otherwise, quadratic effects are aliased into one value if curvature is detected. The curvature detection in the latency model also led to a question of curvature in the accuracy error models. Nevertheless, curvature was unable to be tested without center points in the accuracy error models. Therefore, a definitive screening design should have been used instead of a fractional factorial. The definitive screening design would have allowed a three level design to be performed in few runs, since time was a motive in choosing the fractional factorial design.

The accuracy models from the research led to questioning the effectiveness of the measurement methods. The accuracy error was the intention of the measurements, but one

accuracy method only measured error between GPS and actual distances and the other accuracy method included noise factors in the error. There are two types of error in a leader/follower configuration: targeting error and guiding error. Targeting error is the error of the leader vehicle's position. This error is from where the follower vehicle thinks the leader vehicle is. The guiding error involves the error of the follower vehicle's position. This error is from where the follower vehicle thinks it is at. One possibility of measuring guiding error could be by giving a vehicle "Fly-to-Here" commands and finding how far away from the point the vehicle stops. Finding an accuracy measurement method that identifies guiding error and targeting error could produce a more effective accuracy model. Otherwise, comparing actual distance between the vehicles to the assigned offset may produce a more effective accuracy model.

One way of possibly measuring the GPS error for each vehicle is by putting the same type of autopilots used on both vehicles at an established offset away from each other on a board. By walking the board around a field, the autopilots' GPS receivers will interpret the autopilots' locations. Comparing these autopilot GPS coordinates to the actual offset between the autopilots could reveal the GPS error of each vehicle.

The original accuracy method used in tests one and two was later found to just be measuring the error between the calculated GPS distance between the vehicles and the actual distance. This error wasn't the offset error of the vehicles. The figure eight position accuracy method included latency and other factors into the error. Using a commanded offset versus actual distance measurement would have measured the offset error while factoring out GPS error for the vehicles, latency, and other accuracy error noise factors.

Different factor settings for the optimum latency and accuracy models prevented a single model from being created for both low latency and accuracy. Accuracy should have still been measured with optimum latency settings and vice versa with latency. This would have showed

the effect of one of the measure of performance model's optimum settings on the remaining measure of performance. Unfortunately, time was a limiting factor.

Though Droid Planner 2 was used in experimentation, there was not much experience with the application. The functionality of the application would have been better understood with more experience.

The biggest regret comes from not knowing enough about the functionality of the multi-rotors. The multi-rotors were taken out to Camp Atterbury, IN for heterogeneous vehicle testing. Yet, the multi-rotors didn't perform as expected. The multi-rotors would immediately start landing at its home location once changed to Guided Mode, when another vehicle was connected to a GCS. Troubleshooting took much time away from experimentation. Therefore, the multi-rotors should have been experimented with a few times for familiarity before being brought out for thesis testing. This time spent troubleshooting multi-rotors took away from expanding experimentation with three vehicles instead of only two. Therefore, three vehicle configurations were never performed in the interest of time.

Recommendations for Future Research

The use of multiple vehicle configurations would be another way of expanding cooperative behavior and control. Experimentation can first include a three rover vehicle configuration to validate operation. DOE tests can be performed with the rover configuration to receive optimum factor levels. These optimum factor levels could then be used towards heterogeneous vehicle configurations, starting with two rovers following a multi-rotor. Then the factor levels could be implemented on two multi-rotors following a rover. The altitudes of the multi-rotors would have to be offset to avoid collision. The Python script would also require collision avoidance algorithms to prevent collision with the other follower vehicle.

With only two vehicles used in the research, multi-rotor and rover, a plane could be integrated into further cooperative behavior and control research. The operation of the multi-rotors with the Python script was the first true test of cooperative behavior and control with aerial vehicles. The performance knowledge obtained from the multi-rotors gives increased confidence in the Python script's operation with planes. Planes can fly higher, faster, hold heavier payloads, and withstand higher wind gusts than multi-rotors.

Though latency was able to be reduced to about 2.5 seconds between vehicles, there may be other methods of reducing latency further. For instance, Mavlink was researched early but the method was unable to be executed successfully. Mavlink is a user interface operated at the GCS. It sends messages, such as waypoints and parameter settings, to the vehicle from the GCS. However, it doesn't include the Graphical User Interfaces (GUIs) that Mission Planner or other GCS software does, which could slow down processing speeds [37]. Lowering latency could make the research beneficial to more applications.

Although close-formation flight was concluded infeasible with the latency resulted from the research, reducing latency between vehicles to one tenth of a second, or ten hertz could allow for close-formation flight. Once the Python script is verified with the operation of multiple vehicles, flocking and close-formation algorithms can be integrated into the script. The vehicles can then use the script for close-formation maneuvers.

Vehicle following is the primary application demonstrated with the research. However, the theoretical camera calculations in chapter five were never implemented in experimentation in interest of time. Introducing cameras on the vehicles could offer target identification capabilities. These capabilities could be integrated by programming cooperative behaviors and algorithms into the Python script associated with the research.

Bibliography

[1] B. Tousley, "DARPA Swarm Challenge Program: Critical Technology Review," 2014.

[2] N. Mathews, A. L. Christensen, R. O'Grady and M. Dorigo, "Spatially Targeted Communication and Self-Assembly," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, 2012.

[3] 3DRobotics, "Swarming/Formation-Flying Interface (Beta)," [Online]. Available: http://planner.ardupilot.com/wiki/swarming/. [Accessed 15 July 2014].

[4] D. Jacques, "Approval of AFIT Flight Test Operations Involving Small Unmanned Aircraft Systems," Wright-Patterson Air Force Base, 2012.

[5] Air Force Institue of Technology, "Military Flight Release No. R00199," United States Air Force, Wright-Patterson Air Force Base, 2013.

[6] National Center for Biotechnology Information, "MeSH Database," [Online]. Available: http://www.ncbi.nlm.nih.gov/mesh?term=Cooperative%20Behavior. [Accessed 3 August 2014].

[7] J. Brosig, "Identifying cooperative behavior: some experimental results in a prisoner's dilemma game," *Economic Behavior & Organization*, pp. 275-290, 2002.

[8] A. Stranieri, "Self-organizing flocking in behaviorally heterogeneous swarms," Universite Libre De Bruxelles, Brussels, Belgium, 2011.

[9] C. Hapka, "Flock, Swarm, Throng, Platoon, Gang: What is the difference?," 21 July 2014. [Online]. Available: http://ell.stackexchange.com/questions/29643/flock-swarm-throng-platoon-gang-what-is-the-difference. [Accessed 3 August 2014].

[10] S. A. Songer, "Aerial Networking For The Implementation of Cooperative Control on Small Unmanned Aerial Systems," Air Force Institue of Technology, Wright-Patterson Air Force Base, 2013.

[11] V. Kumar, N. Leonard and A. S. Morse, "Cooperative Control," in *Block Island Workshop on Cooperative Control*, Block Island, RI, 2003.

[12] C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," in *ACM SIGGRPAH*, Anaheim, 1987.

[13] J. Forsyth, "Pygame," DISQUS, [Online]. Available: http://pygame.org/project-Flock-1094-.html. [Accessed 27 July 2014].

[14] J. L. Lambach, "Integrating UAS Flocking Operations With Formation Drag Reduction," Air Force Institue of Technology, Wright-Patterson AFB, OH, 2014.

[15] J. P. Boire, "Autonomous Routing of Unmanned Aerial Vehicle (UAV) Relays to Mimic Optimal Trajectories in Real Time," Air Force Institute of Technology, Wright-Patterson AFB, OH, 2011.

[16] C. E. Booth, "Surveillance Using Multiple Unmanned Aerial Vehicles," Air Force Institute of Technology, Wright-Patterson AFB, OH, 2009.

[17] B. T. Clough, "Metrics, Schmetrics! How The Heck Do You Determine A UAV's Autonomy Anyway?," in *Perfomance Metrics for Intelligent Systems Workshop (PerMIS-02)*, Gaitherburg, MD, 2002.

[18] M. L. Cummings, P. Pina and J. W. Crandall, "A Metric Taxonomy for Supervisory Control of Unmanned Vehicles," Cambridge, MA, 2008.

[19] J. Nielson, "Usability engineering," Academic Press, Cambridge, MA, 1993.

[20] P. Gurfil, "Evaluating UAV Flock Mission Performance Using Dudek's Taxonomy," in *American Control Conference*, Portland, OR, 2005.

[21] 3D Robotics Inc., "3DR Pixhawk," [Online]. Available: https://store.3drobotics.com/products/3dr-pixhawk. [Accessed 17 August 2014].

[22] C. J. Neal, "Feasibility of Onboard Processing of Heuristic Path Planning and Navigation Algorithms within SUAS Autopilot Computational Constraints," Air Force Institute of Technology, Wright-Patterson AFB, OH, 2014.

[23] 3DRobotics Inc., "Using the 3DR Radio for telemetry with APM 2.x and PX4," [Online]. Available: http://plane.ardupilot.com/wiki/common-using-the-3dr-radio-for-telemetry-with-apm-and-px4/#Low_latency_mode. [Accessed 6 January 2015].

[24] Traxxas, "Traxxas E-Max Brushless," [Online]. Available: https://traxxas.com/products/models/electric/39087emaxxbrushless. [Accessed 23 February 2015].

[25] 3D Robotics Inc., "3DR X8+ Copter," [Online]. Available: http://store.3drobotics.com/products/x8-plus/. [Accessed 17 August 2014].

[26] 3D Robotics Inc., "Mission Planner," [Online]. Available: http://planner.ardupilot.com/. [Accessed 17 August 2014].

[27] A. Benemann, "DroidPlanner 2," Androidpit, [Online]. Available: https://fs01.androidpit.info/a/66/db/droidplanner-2-66db06-h900.jpg. [Accessed 6 January 2015].

[28] A. Benemann, "DroidPlanner 2," Google Play Store, 1 December 2014. [Online]. Available: https://play.google.com/store/apps/details?id=org.droidplanner. [Accessed 6 January 2015].

[29] Department of Defense, "DoD Architecture Framework Version 2.0," Department of Defense, 2009.

[30] Sparx Systems Pty Ltd., "Enterprise Architect," Sparx Systems Pty Ltd., [Online]. Available: http://www.sparxsystems.com/products/ea/index.html. [Accessed 24 January 2015].

[31] SAS, "JMP Pro," SAS, [Online]. Available: http://www.jmp.com/software/pro/. [Accessed 23 January 2015].

[32] D. C. Montgomery, Design and Analysis of Experiments Eighth Edition, John Wily & Sons, Inc., 2012.

[33] Minitab, "Lack-of-fit and lack-of-fit tests," Minitab Inc., [Online]. Available: http://support.minitab.com/en-us/minitab/17/topic-library/modeling-statistics/regression-and-correlation/regression-models/lack-of-fit-and-lack-of-

fit-tests/. [Accessed 8 January 2015].

[34] M. Pursifull, "DIY Drones," 1 August 2012. [Online]. Available: http://diydrones.com/group/arducopterusergroup/forum/topics/get-gps-position-every-500ms-seconds. [Accessed 3 January 2015].

[35] M. Driels, Weaponeering: Convential Weapon System Effectiveness, Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2004.

[36] S. J. Comstock, "Development of a Low Latency, High Data Rate, Differential GPS Relative Positioning System For UAV Formation Flight Control," Air Force Institute of Technology, Wright-Patterson AFB, OH, 2006.

[37] Q Ground Control, "MAVLink Micro Air Vehicle Communication Protocol," Q Ground Control, [Online]. Available: http://qgroundcontrol.org/mavlink/start. [Accessed 8 January 2015].

[38] L. Abbott, C. Stillings, C. Phillips and G. Knowlan, "Annex A, Section 3 of Risk Management Plan For The Fleeting Target Technology Demonstrator," Air Force Institute of Technology, Wright-Patterson AFB, OH, 2007.

Appendix A: Traxxas Modified Rover Ground Vehicle Setup

Figure 63. Rover Gain Settings

Table 18. Rover Gain Settings

Parameter	Setting
Steer 2 Servo P	1
Steer 2 Servo I	0.1
Steer 2 Servo D	0.1
Steer 2 Servo INT_MAX	50
L1 Control – Turn Control Period	8
L1 Control – Turn Control Damping	0.9
Speed 2 Throttle P	0.5
Speed 2 Throttle I	0.5
Speed 2 Throttle D	0.5
Speed 2 Throttle INT_MAX	50
Throttle Cruise	33
Throttle Min	0
Throttle Max	100
Throttle FS Value	910
Rover Cruise Speed	3.5
Rover Turn Speed	1
Rover Turn Dist	2
Rover WP Radius	2.625
Sonar Trigger cm	100
Sonar Turn Angle	45
Sonar Turn Time	1
Sonar Debounce	2

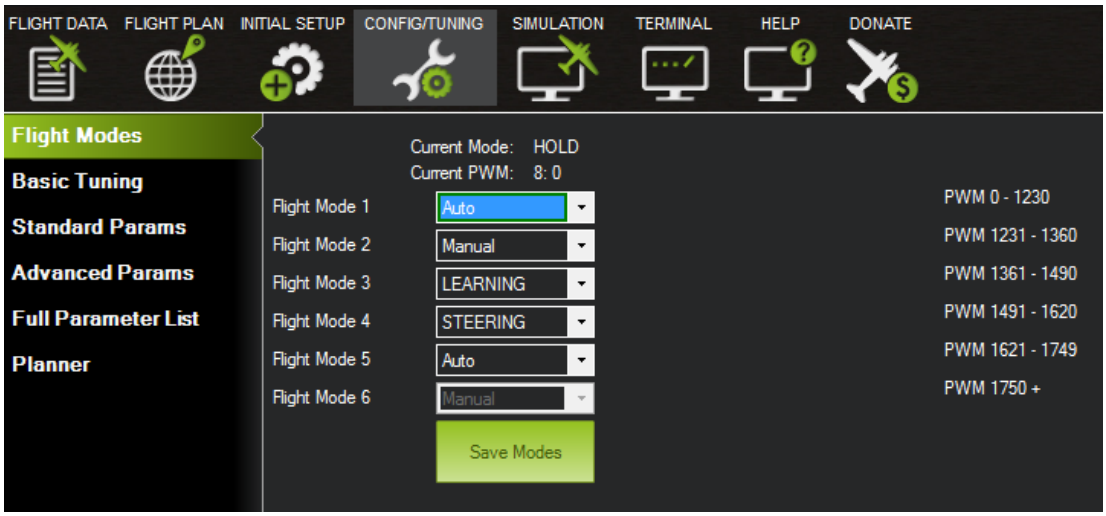

Figure 64. Rover Steering Modes

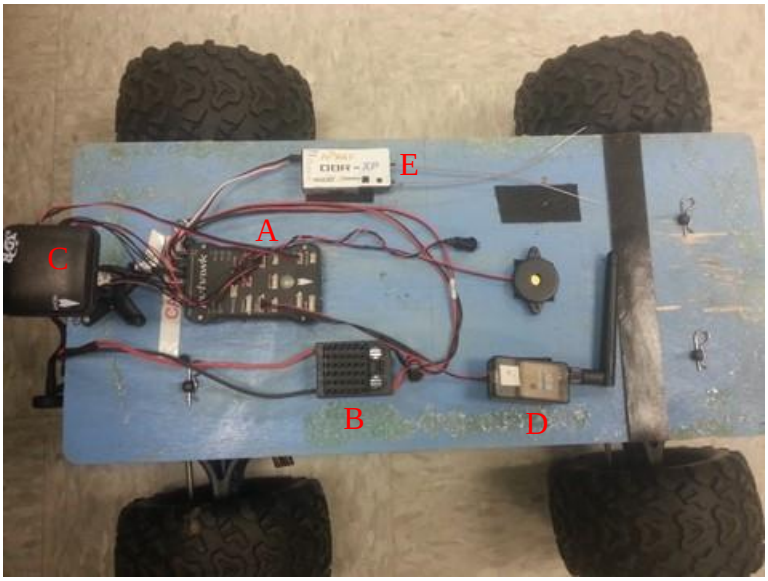

Figure 65. Rover Components

Note: A 5V diode was placed in port 5 of the autopilot to limit the negative affecting current from the servos. Without this diode, the current eventually affected the control of the vehicles.

Figure 66. Traxxas Rover Battery

Table 19. Rover Components

Component	**Description**
Radio Channel	8
Battery	(2) Traxxas NiMH 7-Cell 8.4V 3000mAh
Battery Dimensions	6.10 in x 1.7 in x 0.91 in
Battery Weight	380 g
Battery Connector	Traxxas High-Current Connectors
(A) Autopilot hardware	Pixhawk v2.4.5
Autopilot firmware	ArduRover v2.45
Speed Controller	MXL-6s waterproof electronic
(B) Voltage Regulator	CC BEC PRO: 12S max input
(C) GPS	3DR u-blox GPS with Compass
(D) Ground Station Radio	3DR Radio V2 (915 MHz)
Motors	2200Kv brushless motor
Drive System	Shaft-Driven 4WD
Steering	Bellcrank
Transmission	Single-speed (2nd gear only)
(E) Controller	Turnigy 9X 2.4Ghz Transmitter with FrSky Telemetry Reciever
Frame Type	4-wheel ground vehicle
Tires	6.3” Maxx-Sized tires
Rims	Black-Chrome 3.8” Split-Spoke Wheels
Hex Hubs	17mm
Wheelbase	13.2 in
Overturn Prevention	Adjustable Wheelie Bar
Vehicle Dimensions	22.5 in x 16.5 in x 9.5 in
Center Ground Clearance	4 in
Vehicle Weight with Battery	4.36 kg
Ground Control Station	APM Mission Planner v1.3.7

Table 20. MXL-6s ESC Speed Controller Specifications

Component	**Description**
Input Voltage (cells)	18 NiCad/NiMH 6s LiPo (max: 25.2v)
Case Size	2.2”W x 1.9”D x 1.4”H
Weight	121 g
On-Resistance (@Trans)-FWD/REV	0.0003 ohms per phase

Appendix B: X8 Multi-Rotor Setup

The information directly below is taken directly from: http://store.3drobotics.com/products/x8-plus/?_ga=1.181662884.2037595726.1416447241

"Our 3DR workhorse octocopter is equal to any task. With a flight time of 15 minutes and a payload capacity of over 800 grams, the X8+ is the perfect platform for aerial video.

The X8+ includes

- Controller with live on-screen flight data
- Flight battery and charger
- Operation Manual and Flight Checklist
- Ground station radio with USB and Android adapters

Specs:

- Battery: 4S 14.8V 10,000 mAh 10C
- Battery Dimensions: 6.6 in x 2.6 in x 1.4 in (16.7 cm x 6.5 cm x 3.5 cm)
- Battery Weight: 803 g
- Autopilot hardware: Pixhawk v2.4.5
- Autopilot firmware: ArduCopter 3.2
- GPS: 3DR u-blox GPS with Compass (LEA-6H module, 5 Hz update)
- Ground Station Radio: 3DR Radio v2 (915 MHz or 433 MHz)
- Motors: SunnySky V2216-12 KV800 II (The images above show conical nuts; X8+ ships with hex nuts.)
- Controller: FlySky FS-TH9X with FrSky telemetry module
- Frame Type: X
- Propellers: APC Propeller 11x4.7 SF (4), APC Propeller 11x4.7 SFP (4)
- Vehicle Dimensions: 13.7 in x 20.1 in x 11.8 in (35 cm x 51 cm x 20 cm)
- Payload Capacity: 800 g (1.7 lbs). Additional payload possible up to over 1kg with reduced flight time.
- Vehicle Weight with Battery: 2.56 kg (5.6 lbs)
- Maximum Estimated Flight Time: 15 min

Select from the options below to customize your X8+:

Frequency: Ground station radios allow you to communicate with your aircraft wirelessly in flight. For the US, select 915 MHz. Frequency regulations vary by country, so consult your local airspace communication authority if you're uncertain which frequency is legal in your area.

Extra batteries: The X8+ includes one flight battery. Select this option to extra batteries to your order.

LiveView for GoPro: Select this option to stream live video from a GoPro HERO onto a wireless monitor attached to your X8+ controller. This kit includes a video transmitter, monitor/receiver, cloverleaf antennas, and mounting bracket. Click here for more information.

Extra propellers: The X8+ includes one set of eight propellers. Select this option to add two extra propellers to your order.

Camera gimbal: The Tarot T-2D brushless gimbal uses cutting-edge two-axis stabilization technology to ensure great, stable video in any flight condition. The gimbal comes pre-configured and tuned for a smooth out-of-the-box experience. The kit includes: a pre-assembled Tarot gimbal, a mounting plate, and required cables and hardware.

Case: Select this option to add a travel case for your X8+. Please note that the case ships separately from the X8+, and will fit up to 3 X8+ batteries.

GoPro HERO: Select this option to receive a Go-Pro HERO4+ Black Edition with your X8+! Please note that we cannot ship GoPro internationally. When using a GoPro with X8+, please ensure that the WiFi is turned off; this can cause interference between the X8+ and the controller."

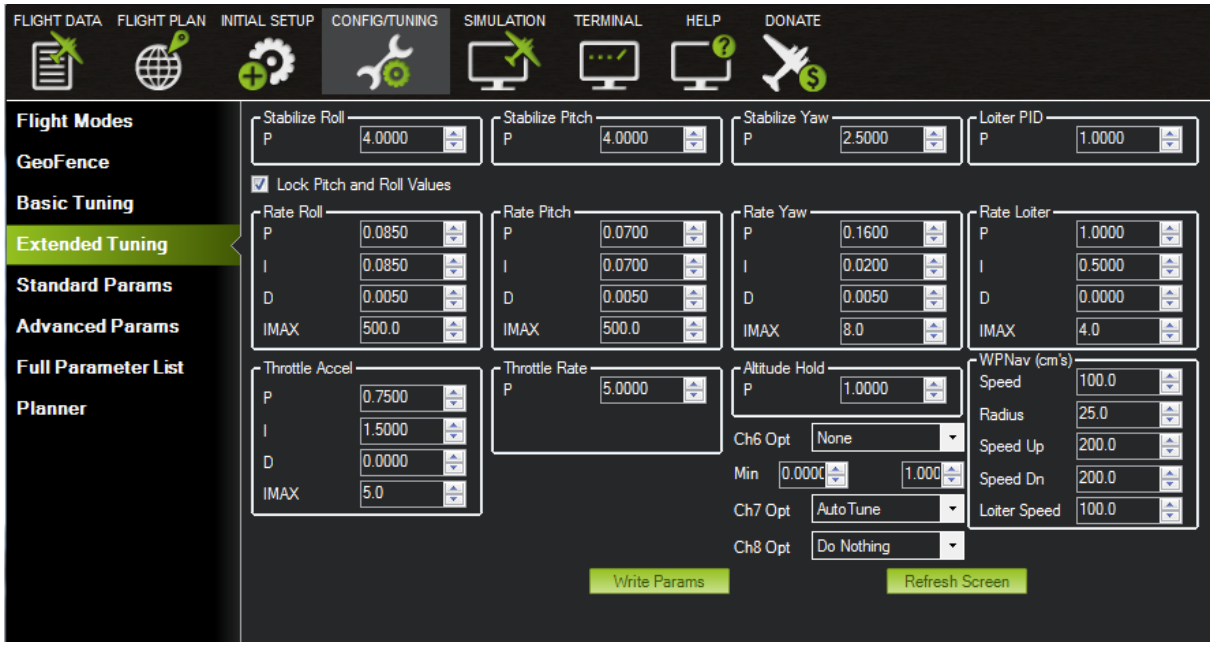

Figure 67. X8 Multi-Rotor Gain Settings

Table 21. X8 Multi-Rotor Gain Settings

Parameter	Setting
Stabilize Roll P	4
Rate Roll P	0.085
Rate Roll I	0.085
Rate Roll D	0.005
Rate Roll IMAX	500
Stabilize Pitch P	4
Rate Pitch P	0.07
Rate Pitch I	0.07
Rate Pitch D	0.005
Rate Pitch IMAX	500
Stabilize Yaw P	2.5
Rate Yaw P	0.16
Rate Yaw I	0.02
Rate Yaw D	0.005
Rate Yaw IMAX	8
Loiter PID P	1
Rate Loiter P	1
Rate Loiter I	0.5
Rate Loiter D	0
Rate Loiter IMAX	4
Throttle Acceleration P	0.75
Throttle Acceleration I	1.5
Throttle Acceleration D	0
Throttle Acceleration IMAX	5
Throttle Rate P	5
Altitude Hold P	1
WPNav (cm's) Speed Up	200
WPNav (cm's) Speed Dn	200
WPNav (cm's) Loiter Speed	100

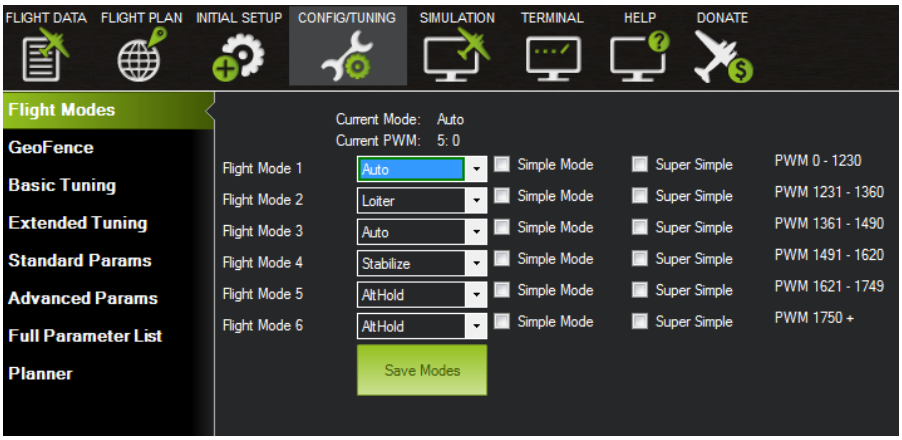

Figure 68. X8 Multi-Rotor Flight Modes

Figure 69. X8 Multi-Rotor Components

Figure 70. Multi-Rotor Battery

Table 22. X8 Multi-Rotor Components

Component	Description
Radio Channel	5
Battery	Tiger Power Atomic-Platinum 4S 14.8V 6000 mAh 35C
Battery Dimensions	16 cm x 5 cm x 4 cm
Battery Weight	680 g
Battery Connector	3DR Power Module with XT60 connector
Autopilot hardware	Pixhawk v2.4.5
Autopilot firmware	ArduCopter v3.1.4
Speed Controller	20 Amp ESCs with SimonK firmware
GPS	3DR u-blox GPS with magnetometer
Ground Station Radio	3DR Radio V2 (915 MHz)
Motors	880 Kv brushless motors
Controller	FrSky 2.4 GHz ACCST Taranis x9D with FrSky telemetry module
Frame Type	X
Propellers	APC Propeller 10x4.7 SF (4), APC Propeller 10x4.7 SFP (4)
Vehicle Dimensions	13.7 in x 20.1 in x 11.8 in
Payload Capacity	800 g
Vehicle Weight with Battery	2.45 kg
Maximum Estimated Flight Time	12-13 min
Ground Control Station 1	APM Mission Planner v1.3.7
Ground Control Station 2	Droid Planner 2_v2.8.6_RC3

Appendix C: Leader Vehicle Python Script

```
#-------------------------------------------------------------------------------

# Name:        Leader Vehicle Python Script
# Purpose:     UDP server on Mission Planner
#
# Author:      AUSTIN & DR. JOHN COLOMBI
# Created:     13/03/2013
# Copyright:   (c) AUSTIN 2013
# Modified:     STEFAN HARDY
#-------------------------------------------------------------------------------

#import libraries for commands or functions used
import socket
import sys
import math
import clr
import time
import re, string
clr.AddReference("MissionPlanner")
import MissionPlanner
clr.AddReference("MissionPlanner.Utilities") # includes the Utilities class
from MissionPlanner.Utilities import Locationwp

HOST = ''   # Symbolic name meaning all available interfaces
SPORT = 4000 # Arbitrary non-privileged port

REMOTE = '192.168.3.4' #IP address of follower vehicle GCS connecting to. Use
#'localhost' if on same GCS.
# Datagram (udp) socket

ssock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #Creates a send socket
#for connection between GCSs
print 'Sockets created'

address = (REMOTE, SPORT) #contains IP of follower vehicle and port number
#infinite loop that sends out current lat,long,heading,and alt parameters of
#leader vehicle
while 1:
        lat = str(cs.lat) #Converts current state (cs) latitude of leader
        #vehicle to string.
        lng = str(cs.lng) #Converts current state (cs) longitude of leader
        #vehicle to string.
        heading = str(cs.yaw) #Converts current state (cs) yaw of leader
        #vehicle to string.
        alt = str(cs.alt) #Converts current state (cs) altitude of leader
        #vehicle to string.
        #List of parameters (cs.?) able to be retrieved from Mission Planner
        #can be found at:
        #http://copter.ardupilot.com/wiki/common-using-python-scripts-in-
        #mission-planner/

        #Ties current leader vehicle parameters to msg for sending to follower
        #vehicle GCS.
        msg = lat + ' ' + lng + ' ' + heading + ' ' + alt
```

```
54.
55.         #prints lat, lng, heading, and alt in command window
56.         print lat
57.         print lng
58.         print heading
59.         print alt
60.         Script.Sleep(500) #Socket read waiting (set delay) in miliseconds.
61.         #Default is 1000 ms.
62.         ssock.sendto(msg,address) #Send msg to address (Follower GCS)
63.         print 'sent data'
64.         print time.strftime('%X %x %Z') #Local computer time stamp in command
65.         #window.
66.
67. # exit
68. rsock.close() #closes socket
69. print 'Script End'
```

Appendix D: Rover Follower Distance Offset Python Script

```
#-------------------------------------------------------------------------------

# Name:        Rover Follower Vehicle Distance Offset w/ Leader Vehicle
# Purpose:     UDP client on python
#
# Author:      AUSTIN & DR. JOHN COLOMBI
# Created:     13/03/2013
# Copyright:   (c) AUSTIN 2013
# Modified:     STEFAN HARDY
#-------------------------------------------------------------------------------

#import libraries for commands or functions used
import socket
import sys
import math
from math import sqrt
import clr
import time
import re, string
clr.AddReference("MissionPlanner.Utilities")
import MissionPlanner #import *
clr.AddReference("MissionPlanner.Utilities") #includes the Utilities class
from MissionPlanner.Utilities import Locationwp

HOST = '192.168.3.4'   #IP address of Ground Control Station (GCS) of
#Follower Vehicle. Use 'localhost' if on same GCS.
RPORT = 4000 # Arbitrary non-privileged port

REMOTE = ''
# Datagram (udp) socket

rsock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) #Creates a receive
#socket for connection between GCSs
print 'Sockets created'

# Bind socket to local host and port
try:
   rsock.bind((HOST,RPORT)) #Attempts to bind socket to host, or follower GCS,
   #and RPORT
except socket.error, msg: #If not bound, prints error message
   #print 'Bind failed. Error Code:'
   sys.stderr.write("[ERROR] %s\n" % msg[1])
   rsock.close()
   sys.exit()

print 'Receive Socket bind complete on ' + str(RPORT)

print 'Starting Follow' #Prints "Starting Follow" on the command window
Script.ChangeMode("Guided") #Changes follower vehicle mode to "Guided" in
#Mission Planner
print 'Guided Mode'

#keep talking with the Mission Planner server
while 1:
```

```
54.     # receive data from server (data, addr)
55.     msg = rsock.recv(1024) #recieves msg, containing leader vehicle coordinates

56.     pattern = re.compile("[ ]") #Marks at what points in msg to split up msg
57.     parameters = pattern.split(msg) #Splits msg at points where pattern exist in
        #msg
58.
59.     #leader vehicle coordinates are below
60.     latData = parameters[0] #first parameter in split msg is latitude
61.     lngData = parameters[1] #Second parameter in split msg is longitude
62.     headingData = parameters[2] #Third parameter in split msg is heading
63.     altData = parameters[3] #Last parameter in split msg is altitude
64.
65.     #Must convert all parameters to float for calculations
66.     float_lat = float(latData)
67.     float_lng = float(lngData)
68.     float_heading = float(headingData)
69.     float_alt = float(altData)
70.
71.     #Calculations for follower vehicle geodetic offset are made below based
72.     #off of leader vehicle's geodetic location.
73.     #These follower vehicle waypoint calculations are repeated through a loop
74.     #until script is manually stopped.
75.     """Follower Offset"""
76.     XOffset= 1 #User Input, in meters, for x axis offset
77.     YOffset= 0 #User Input, in meters, for y axis offset
78.     brng = math.radians(270) #User input heading angle offset of follower in
79.     #relation to leader.  0 degrees is forward.  Converts heading to radians
80.
81.     XOffset = float(XOffset)/10 #XOffset seems to be in decameters or 10 meters.
82.     #This converts it to meters
83.     YOffset = float(YOffset)/10 #YOffset seems to be in decameters or 10 meters.
84.     #This converts it to meters
85.     R = 637100 #Radius of the Earth in m
86.     d = math.sqrt((XOffset**2)+(YOffset**2)) #Distance in m. ** is exponent
87.
88.     print d #Prints calculated follower vehicle offset hypotenuse, or distance
89.     #from leader to follower, to command window
90.     lat1 = math.radians(float_lat) #Current leader lat point converted to
91.     #radians
92.     lon1 = math.radians(float_lng) #Current leader long point converted to
93.     #radians
94.
95.     lat2 = math.asin( math.sin(lat1)*math.cos(d/R) + math.cos(lat1)*math.sin(d/R
    )*math.cos(brng))
96.     #Latitude position of follower from offset
97.     lon2 = lon1 + math.atan2(math.sin(brng)*math.sin(d/R)*math.cos(lat1), math.c
    os(d/R)-math.sin(lat1)*math.sin(lat2))
98.     #Longitude position of follower from offset
99.
100.            lat2 = math.degrees(lat2) #Converts follower latitude to degrees
101.            lon2 = math.degrees(lon2) #Converts follower longitude to degrees
102.
103.            #Converts follower vehicle offset coordinates to float for waypoint
104.            #writing
105.            float_lat = float(lat2)
106.            float_lng = float(lon2)
107.
108.            #Prints follower vehicle offset coordinates to command window
```

```
            print(lat2)
            print(lon2)

            """Writing Waypoints"""
            item = MissionPlanner.Utilities.Locationwp() # creating waypoint
            MissionPlanner.Utilities.Locationwp.lat.SetValue(item,float_lat)
            #Writes follower vehicle latitude coordinate as waypoint in
            #Mission Planner
            MissionPlanner.Utilities.Locationwp.lng.SetValue(item,float_lng)
            #Writes follower vehicle longitude coordinate as waypoint in Mission
            #Planner
            MissionPlanner.Utilities.Locationwp.alt.SetValue(item,float_alt)
            #Writes follower vehicle altitude coordinate as waypoint in
            #Mission Planner
            #Can only use lat,lng, or alt for waypoint writing
            #MUST WRITE ALL THREE COORDINATES TO WRITE A WAYPOINT OR WAYPOINT
            #WILL NEVER BE SUCCESSFULLY RECOGNIZED! Will just continue to loop
            #with 0 latency.
            MAV.setGuidedModeWP(item) #sets waypoint.  The largest latency will
            #be recognized from this line. Must go through Mission Planner
            #to set #waypoint.

            print 'Waypoint Sent'
            print time.strftime('%X %x %Z') #Prints time on computer in command
            #Window.
            #Used to show latency between leader and follower GCS
        # exit
        rsock.close() #closes socket between GCS
        print 'Script End'
```

Appendix E: Rover Follower Vehicle Heading Offset Python Script

```
#-------------------------------------------------------------------------------

# Name:         Rover Follower Vehicle w/ Leader Vehicle heading offset
# Purpose:      UDP client on python
#
# Author:       AUSTIN & DR. JOHN COLOMBI
# Created:      13/03/2013
# Copyright:    (c) AUSTIN 2013
# Modified:      STEFAN HARDY
#-------------------------------------------------------------------------------

#import libraries for commands or functions used
import socket
import sys
import math
from math import sqrt
import clr
import time
import re, string
clr.AddReference("MissionPlanner.Utilities")
import MissionPlanner #import *
clr.AddReference("MissionPlanner.Utilities") #includes the Utilities class
from MissionPlanner.Utilities import Locationwp

HOST = '192.168.3.2' #IP address of Ground Control Station (GCS) of
#Follower Vehicle. Use 'localhost' if on same GCS.
RPORT = 4000 # Arbitrary non-privileged port

REMOTE = ''
# Datagram (udp) socket


rsock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) #Creates a receive
#socket for connection between GCSs
print 'Sockets created'

# Bind socket to local host and port
try:
   rsock.bind((HOST,RPORT)) #Attempts to bind socket to host, or follower
   #GCS, and RPORT
except socket.error, msg: #If not bound, prints error message
   #print 'Bind failed. Error Code:'
   sys.stderr.write("[ERROR] %s\n" % msg[1])
   rsock.close()
   sys.exit()

print 'Receive Socket bind complete on ' + str(RPORT)

print 'Starting Follow' #Prints "Starting Follow" on the command window
Script.ChangeMode("Guided") #Changes follower vehicle mode to "Guided" in
#Mission Planner
print 'Guided Mode'
```

```
52.
53. #keep talking with the Mission Planner server
54. while 1:
55.     # receive data from server (data, addr)
56.     msg = rsock.recv(1024) #recieves msg, containing leader vehicle
57.     #coordinates
58.     pattern = re.compile("[ ]") #Marks at what points in msg to split
59.     #up msg
60.     parameters = pattern.split(msg) #Splits msg at points where pattern
61.     #exist in msg
62.
63.     #leader vehicle coordinates are below
64.     latData = parameters[0] #first parameter in split msg is latitude
65.     lngData = parameters[1] #Second parameter in split msg is longitude
66.     headingData = parameters[2] #Third parameter in split msg is heading
67.     altData = parameters[3] #Last parameter in split msg is altitude
68.
69.     #Must convert all parameters to float for calculations
70.     float_lat = float(latData)
71.     float_lng = float(lngData)
72.     float_heading = float(headingData)
73.     float_alt = float(altData)
74.
75.
76.     """Safety Manual Mode Switch"""
77.     #When safety pilot radio control is switched to Manual, ch8in climbs
78.     #above 1700
79.     #If ch5in of follower vehicle climbs above 1700, script closes
80.     if cs.ch8in > 1700:
81.        #print cs.mode
82.        Script.ChangeMode("Manual") #Changes mode of follower vehicle to
83.        #Stabilize in Mission Planner
84.        print cs.mode
85.        print cs.ch8in
86.        rsock.close() #closes socket between GCSs
87.        sys.exit() #Ends script
88.     else:
89.
90.        #Else, calculations for follower vehicle offset are made below
91.        #based off of leader vehicle's heading.
92.        #These follower vehicle waypoint calculations are repeated through a
93.        #loop until script is manually stopped or safety switch is triggered.
94.        """Follower Offset"""
95.        XOffset= float(0) #User Input for x axis offset
96.        YOffset= float(0) #User Input for y axis offset
97.        brng = math.radians(float_heading) #User input heading angle of
98.        #follower in relation to leader.  0 degrees is forward.
99.
100.              d = math.sqrt((XOffset**2)+(YOffset**2)) #Distance in m
101.
102.              MperLat = 69.172*1609.34 #meters per degree of latitude. Length
103.              #of degree (miles) at equator * meters in a mile
104.              MperLong = math.cos(float_lat)*69.172*1609.34 #meters per degree
105.              #of longitude
106.
107.              Lat_Offset_meters = YOffset/MperLat #lat distance offset in
108.              #meters
109.              Long_Offset_meters = XOffset/MperLong #long distance offset in
110.              #meters
```

```
111.
112.                 #Follower vehicle waypoint coordinate calculations in relation to
113.                 #heading of leader vehicle
114.                 Follower_lat = float_lat + (Long_Offset_meters*math.sin(brng)) +
    (Lat_Offset_meters*math.cos(brng))
115.                 #rotates lat follower offset in relation to heading of leader
116.                 Follower_long = float_lng -
     (Long_Offset_meters*math.cos(brng)) + (Lat_Offset_meters*math.sin(brng))
117.                 #rotates long follower offset in relation to heading of leader
118.                 Follower_alt = 10 #set constant altitude of follower vehicle, in
119.                 #meters. Altitude must be set regardless of ground or air vehicle

120.                 #follower vehicle waypoint coordinates are converted to float,
121.                 #just in case
122.                 float_lat = float(Follower_lat)
123.                 float_lng = float(Follower_long)
124.                 float_alt = float(Follower_alt)
125.
126.                 #Prints out the follower vehicle waypoint coordinates in the
127.                 #command window
128.                 print(float_lat)
129.                 print(float_lng)
130.                 print(float_heading)
131.                 print(float_alt)
132.
133.                 """Writing Waypoints"""
134.                 item = MissionPlanner.Utilities.Locationwp() # creating waypoint

135.                 MissionPlanner.Utilities.Locationwp.lat.SetValue(item,float_lat)
136.                 #Writes follower vehicle latitude coordinate as waypoint in
137.                 #Mission Planner
138.                 MissionPlanner.Utilities.Locationwp.lng.SetValue(item,float_lng)
139.                 #Writes follower vehicle longitude coordinate as waypoint in
140.                 #Mission Planner
141.                 MissionPlanner.Utilities.Locationwp.alt.SetValue(item,float_alt)
142.                 #Writes follower vehicle altitude coordinate as waypoint in
143.                 #Mission Planner
144.                 #Can only use lat,lng, or alt for waypoint writing
145.                 #MUST WRITE ALL THREE COORDINATES TO WRITE A WAYPOINT OR WAYPOINT
146.                 #WILL NEVER BE SUCCESSFULLY RECOGNIZED! Will just continue to
147.                 #loop with 0 latency.
148.                 MAV.setGuidedModeWP(item) #sets waypoint.  The largest latency
149.                 #will be recognized from this line. Must go through Mission
150.                 #Planner to set
151.                 #waypoint.
152.                 print 'Waypoint Sent'
153.                 print time.strftime('%X %x %Z') #Prints time on computer in
154.                 #command window.  Used to show latency between leader
155.                 #and follower GCS
156.         # exit
157.         rsock.close()
158.         print 'Script End'
```

Appendix F: Multi-Rotor Follower Vehicle Heading Offset Python Script

```
1.  #--------------------------------------------------------------------------------
2.  # Name:         Multi-Rotor Follower Vehicle w/ Leader Vehicle heading offset
3.  # Purpose:      UDP client on python
4.  #
5.  # Author:       AUSTIN & DR. JOHN COLOMBI
6.  # Created:      13/03/2013
7.  # Copyright:    (c) AUSTIN 2013
8.  # Modified:      STEFAN HARDY
9.  #--------------------------------------------------------------------------------
10.
11. #import libraries for commands or functions used
12. import socket
13. import sys
14. import math
15. from math import sqrt
16. import clr
17. import time
18. import re, string
19. clr.AddReference("MissionPlanner.Utilities")
20. import MissionPlanner #import *
21. clr.AddReference("MissionPlanner.Utilities") #includes the Utilities class
22. from MissionPlanner.Utilities import Locationwp
23.
24. HOST = '192.168.3.4' #IP address of Ground Control Station (GCS) of
25. #Follower Vehicle. Use 'localhost' if on same GCS.
26. RPORT = 4000 # Arbitrary non-privileged port
27.
28. REMOTE = ''
29. # Datagram (udp) socket
30.
31. print 'Starting Follow' #Prints "Starting Follow" on the command window
32. Script.ChangeMode("Guided")# changes follower vehicle mode to "Guided"
33. #in Mission Planner
34. print 'Guided Mode'
35.
36. rsock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM) #Creates a
37. #receive socket for connection between GCSs
38. print 'Sockets created'
39.
40. # Bind socket to host and port
41. try:
42.    rsock.bind((HOST,RPORT)) #Attempts to bind socket to host, or follower
43.    #GCS, and RPORT
44. except socket.error, msg: #If not bound, prints error message
45.    #print 'Bind failed. Error Code:'
46.    sys.stderr.write("[ERROR] %s\n" % msg[1])
47.    rsock.close()
48.    sys.exit()
49.
50. print 'Receive Socket bind complete on ' + str(RPORT)
51.
52.
53.
```

```
#keep talking with the Mission Planner server
while 1:
    # receive data from server (data, addr)
    msg = rsock.recv(1024) #recieves msg, containing leader vehicle
    #coordinates
    pattern = re.compile("[ ]") #Marks at what points in msg to split
    #up msg
    parameters = pattern.split(msg) #Splits msg at points where pattern
    #exist in msg

    #leader vehicle coordinates are below
    latData = parameters[0] #first parameter in split msg is latitude
    lngData = parameters[1] #Second parameter in split msg is longitude
    headingData = parameters[2] #Third parameter in split msg is heading
    altData = parameters[3] #Last parameter in split msg is altitude

    #Must convert all parameters to float for calculations
    float_lat = float(latData)
    float_lng = float(lngData)
    float_heading = float(headingData)
    float_alt = float(altData)

    """Safety Manual Mode Switch"""
   #When safety pilot radio control is switched to Stabilize, ch8in
    #remains between 1400-1900
   #If ch5in of follower vehicle falls in 1400-1900 window, script closes
   if cs.ch5in > 1400 and cs.ch5in < 1900:
       Script.ChangeMode("Stabilize") #Changes mode of follower vehicle
       #to Stabilize in Mission Planner
       print cs.mode
       print cs.ch5in
       rsock.close() #closes socket between GCSs
       sys.exit() #Ends script
    else:

       #Else, calculations for follower vehicle offset are made below
       #based off of leader vehicle's heading.
       #These follower vehicle waypoint calculations are repeated through a
       #loop until script is manually stopped or safety switch is triggered.
       """Follower Offset"""
       XOffset= float(0) #User Input for x axis offset
       YOffset= float(0) #User Input for y axis offset
       brng = math.radians(float_heading)#User input heading angle of
       #follower in relation to leader.  0 degrees is forward.

       d = math.sqrt((XOffset**2)+(YOffset**2)) #Distance in m

              MperLat = 69.172*1609.34 #meters per degree of latitude. Length
              #of degree (miles) at equator * meters in a mile
              MperLong = math.cos(float_lat)*69.172*1609.34 #meters per degree
              #of longitude

              Lat_Offset_meters = YOffset/MperLat #lat distance offset in
              #meters
              Long_Offset_meters = XOffset/MperLong #long distance offset in
              #meters

              #Follower vehicle waypoint coordinate calculations in relation
              #to heading of leader vehicle
```

```
113.                Follower_lat = float_lat + (Long_Offset_meters*math.sin(brng)) +
    (Lat_Offset_meters*math.cos(brng))
114.                #rotates lat follower offset in relation to heading of leader
115.                Follower_long = float_lng -
     (Long_Offset_meters*math.cos(brng)) + (Lat_Offset_meters*math.sin(brng))
116.                #rotates long follower offset in relation to heading of leader
117.                Follower_alt = 10 #set constant altitude of follower vehicle,
118.                #in meters
119.
120.                #follower vehicle waypoint coordinates are converted to float,
121.                #just in case
122.                float_lat = float(Follower_lat)
123.                float_lng = float(Follower_long)
124.                float_alt = float(Follower_alt)
125.
126.                #Prints out the follower vehicle waypoint coordinates in the
127.                #command window
128.                print(float_lat)
129.                print(float_lng)
130.                print(float_heading)
131.                print(float_alt)
132.
133.                """Writing Waypoints"""
134.                item = MissionPlanner.Utilities.Locationwp() # creating waypoint

135.                MissionPlanner.Utilities.Locationwp.lat.SetValue(item,float_lat)

136.                #Writes follower vehicle latitude coordinate as waypoint in
137.                #Mission Planner
138.                MissionPlanner.Utilities.Locationwp.lng.SetValue(item,float_lng)

139.                #Writes follower vehicle longitude coordinate as waypoint in
140.                #Mission Planner
141.                MissionPlanner.Utilities.Locationwp.alt.SetValue(item,float_alt)

142.                #Writes follower vehicle altitude coordinate as waypoint in
143.                #Mission Planner
144.                #Can only use lat,lng, or alt for waypoint writing
145.                #MUST WRITE ALL THREE COORDINATES TO WRITE A WAYPOINT OR WAYPOINT

146.                #WILL NEVER BE SUCCESSFULLY RECOGNIZED! Will just continue to
147.                #loop with 0 latency.
148.                MAV.setGuidedModeWP(item) #sets waypoint.  The largest latency
149.                #will be recognized from this line. Must go through
150.                #Mission Planner to set waypoint.
151.
152.                #Prints out set waypoints through Mission Planner
153.                print MissionPlanner.Utilities.Locationwp.lat.SetValue(item,float
    _lat)
154.                print MissionPlanner.Utilities.Locationwp.lng.SetValue(item,float
    _lng)
155.                print MissionPlanner.Utilities.Locationwp.alt.SetValue(item,float
    _alt)
156.
157.                print 'Waypoint Sent'
158.                print time.strftime('%X %x %Z') #Prints time on computer in
159.                #command window.  Used to show latency between leader and
160.                #follower GCS
161.
```

```
162.        rsock.close() #closes socket between GCS
163.        print 'Script End'
```

Appendix G: AFIT Document 5028 Test Project Technical and Safety Review

TEST PROJECT TECHNICAL AND SAFETY REVIEW

SECTION I	PROJECT INFORMATION			
Test Project Title		Overall Risk	Control #	Test Dept
Co-op Behavior & Control w/ Heterogeneous Vehicles –Thesis		LOW	14-04	ENV

Typed Name and Grade	Signature	Email Address	Phone Number	Date
Principal Investigator				
Dr. David Jacques		david.jacques@afit.edu	X3329	22OCT2014
Project Safety Lead				
Stefan Hardy, 1st Lt		stefan.hardy@afit.edu		22OCT 2014

SECTION II	TECHNICAL/SAFETY REVIEW				
Typed Name and Grade	Position	Signature	Date	Coord	
AFIT Flight Test Safety Officer				Yes	No
Jeremy Agte, Lt Col	AFIT/ENY		22OCT 2014		
Safety Reviewer #1					
Jason Freels, Maj	AFIT/ENV		22OCT 2014		
Safety Reviewer #2					
Mathew Dillsaver, Maj	AFIT/ENY		22OCT 2014		

SECTION III	COORDINATING COMMENTS
(Reviewer should initial next to any comments)	

AFIT Document 5028, Apr 2013. Previous editions will not be used.

PROJECT DESCRIPTION – AFIT CO-OP BEHAVIOR & CONTROL W/ HETEROGENEOUS VEHICLES

1. BACKGROUND

   a. Mission Title: Autonomous leader/follower behavior between multi-rotors, and between rovers (trucks) and multi-rotors

   b. Description:
   This test will utilize 3DR X8 multi-rotor small unmanned aerial systems (SUAS) (Group I) together and with rover (trucks) vehicles. Python, a programming language, will be used to force leader/follower behavior and relationships between the multi-rotors and rovers. A combination of manual control and assigned waypoints (AUTO) will be established with the multi-rotors. The tests will measure the latency between the leader and follower vehicles, as well as the accuracy of the position offset of the follower in relation to the leader, through a set of controlled parameters involving Waypoint Radius, Cruise Speed, Telemetry Rate, Max Window (3DR Radios), and Sleep Time (Python).

   c. Purpose:
      i. The main objective of the flight test is to determine what factor settings from Mission Planner, the 3DR radios, and Python will achieve the lowest latency and accuracy error of the follower vehicles. The same optimum parameter settings found on the rovers, using Design of Experiments (DOE), will be used on the multi-rotors to determine the effects of the settings on latency and accuracy.

   d. List of AFIT and non-AFIT assets at risk:

      i. 3DR X8 multi-rotor small UAS
      ii. Rovers (trucks)
      iii. AFIT Personnel (a mix of several military and civilian staff and students)
      iv. A vehicle and trailer owned and operated by CESI (AFIT support contractor)
      v. Support building around the Himsel Army Airfield (AAF)
      vi. Any personnel within a ½ mile radius of Himsel AAF(for standard test operations, see 4.c.vi for maximum range footprint)

   e. Location of test:
   Himsel Army Airfield, Camp Atterbury Joint Maneuver Training Center, IN
   UAS Airstrip, Camp Atterbury Joint Maneuver Training Center, IN

f. Planned dates of the test:
28 – 30 October 2014

g. Number of projected flights during the test period:
Approximately 8 flights

2. MISHAP RESPONSIBILITIES

a. Should an incident occur in which one of the UAVs is damaged or destroyed, the AFIT Flight Test Safety Officer (FTSO) will be notified via the After Action Report (Section VII of this document).
b. If an incident occurs in which property owned by the Army, Camp Atterbury or civilians is damaged and/or any personnel are injured, the Camp Atterbury Safety Office will be notified immediately. That office will make a determination on whether or not to initiate an investigation. In addition, the AFIT Safety Office will be notified within 5 working days per AFIT's Mishap Notification Procedures. If an injury or illness results in lost duty time or hospitalization, then the AFIT Safety Office will be notified immediately. The Principal Investigator will be responsible for submitting any of the required mishap reports as defined in AFIT's Mishap Notification Procedures. For further information, refer to the Mishap Notification Procedures posted in the Safety folder under the 'Mishap Reporting' tab on the AFIT Intranet site.

3. TEST OBJECTIVES

Summarize the top-level objectives listed in the test plan.

a. Objective 1 – Set one multi-rotor as leader and one rover as follower, and use DOE optimum parameter settings to find effect on latency.
b. Objective 2 – Set one multi-rotor as leader and one rover as follower, and use DOE optimum parameter settings to find effect on accuracy.
c. Objective 3 – Set one rover as leader and one multi-rotor as follower, and use DOE optimum parameter settings to find effect on latency.
d. Objective 4 – Set one rover as leader and one multi-rotor as follower, and use DOE optimum parameter settings to find effect on accuracy.
e. Objective 5 – Set one multi-rotor as leader and other as follower, and use DOE optimum parameter settings to find effect on latency.
f. Objective 6 – Set one multi-rotor as leader and other as follower, and use DOE optimum parameter settings to find effect on accuracy.
g. Objective 7 – Incorporate 3rd vehicle (multi-rotor) as follower in leader/follower relationship. Run optimum DOE parameter setting for latency effect
h. Objective 8 – Incorporate 3rd vehicle (multi-rotor) as follower in leader/follower relationship. Run optimum DOE parameter setting for accuracy effect.

4. TEST ITEM DESCRIPTION

a. Manufacturer: 3D Robotics
b. Model: RTF X8
c. Characteristics : 24 in x 24 in x 8 in, Flying Weight 5.4 lbs (w/ battery)
d. Power Plant: 880 Kv brushless motors with 10x4.7 propellers.
e. Avionics: Pixhawk Autopilot
f. Datalink:
    i. Autopilot – 3DRobotics 915 MHz FHSS modems;
    ii. Safety Pilot RC Control – Spektrum DX18 2.4 GHz Tx with Spektrum AR12020 2.4 GHz Rx.
g. Method of pilotage: Manual pilot control for takeoff and landings. First flight Autopilot control when AGL altitude exceeds 10 feet through Python script. Autopilot commands are provided by ground station or onboard computer. Pilot can take manual control at any time during operations. If communications are lost with autopilot, autopilot will fly to rally point for manual recovery by backup R/C system.
h. Flight Altitude: Maximum altitude of 20 feet AGL with normal operating altitudes of 10 – 15 feet AGL.
i. Range: Continuous Line-of-Sight (LOS) distances only. Maximum range of autopilot/ ground station radio link is about 6.2 miles (10 km). Maximum range of R/C radio link has been tested to 1 mile. Maximum duration of flight with full battery is approximately 12-13 minutes.
j. Wind Speed: For launch/landing operations, a maximum wind speed (including gusts) less than 10 knots, with a cross wind of less than 10 knots.
k. Launch Method: Manual pilot control via R/C pilot radio. Both pilot and aircraft handler will maintain positive communication and ensure the aircraft is free from obstructions.
l. Landing Method: Manual pilot control via R/C pilot console smooth runway (grass/pavement/gravel) and free of obstructions.
m. Flight Control: Ground control station (GCS) control through COTS autopilot, mechanically-linked servos for the model aircraft's control surfaces including throttle. A backup system using a COTS R/C transmitter will control same control surfaces and propulsion motor in the event of autopilot failure.
    i. Autopilot: The autopilot system consists of on-board avionics and a ground station, communicating using the 902 – 928 MHz band with 100 mW of RF power. The COTS vendor supplies software for the GCS. Through this software, waypoints can be entered over a geo-referenced map, with same map displaying the GPS location of the UAV. Mission altitude limits are established beforehand to ensure that avionics will keep the UAV at a safe altitude if an erroneous altitude is entered into a waypoint.
    ii. Manual: Manual control is executed by the R/C safety pilot for takeoff, landing, and in the event that unsafe flight conditions are encountered

while under autopilot control. This is done through a COTS R/C transmitter and receiver system operating the same mechanical servos and linkages.

n. Failure Modes:
    i. Lost Communication Link – If communications are lost for more than 20 seconds, the vehicle enters return-to-launch mode. If communications are reestablished, the vehicle can be commanded to resume the normal flight path. If communications are never reestablished, the safety pilot may use the RC link to land the aircraft under manual control.
    ii. Lost GPS – If the aircraft loses GPS it will enter a hover in place until GPS is recovered. The ground station audibly notifies the operator that GPS is lost, and at this time the safety pilot would assume manual control of the aircraft if GPS is not reacquired.
    iii. Unresponsive Flight Controls - Visual detection will be used to identify aircraft problems. If aircraft cannot be controlled and safely returned to the landing site, the motor will be shut down by the operator and the aircraft will crash land in its current vicinity. There is no servo redundancy.
    iv. Loss of Propulsion – Should the aircraft lose propulsion, the aircraft will fall to the ground. Therefore a low altitude will be maintained to prevent vehicle destruction. The X8 has eight propellers (2 on each leg) that will serve as backups if one of the primary propellers fail, as a safety procedure. In this instance, the operator has the ability to take control and guide the aircraft to the landing site.
    v. Loss of Autopilot – If the autopilot fails to function, this will typically result in loss of power to servos. The RC transmitter will be placed in manual mode, throttle down, with all control surfaces centered.
    vi. Loss of Ground Control Station – A gas powered generator supplies AC power to all ground station components. The autopilot ground station has internal lithium batteries as a backup power source. If all independent sources of ground station electrical power are lost, the communications link will be cut, and the vehicle will fly to its “Lost Link” rally point where battery-operated R/C control will be established for landing.

a. Describe the test facilities to be used:
    i. The Himsel AAF is a fully functional airfield located on Army property and under restricted airspace. The field has a single north/south runway. The field is located in an isolated area of the base adjacent to the weapons range. The field is controlled and flight operations will always be cleared by the Himsel tower controller. The airfield operations building is located at the north end of the runway and has restroom facilities.

ii. The UAS strip contains a shed and a long gravel road in parallel with the runway. The strip is a paved north/south runway in an open area.

iii. Yellow bounding boxes show anticipated flight areas to meet test objectives. Left box is Himsel AAF and right box is the UAS strip.

SYSTEM MATURITY

b. Describe testing that supports readiness:

The X8 multi-rotor is an off-the-shelf hobbyist R/C multi-rotor and has been owned and flown by hobbyists around the world. With the intent of hosting research payloads (sensors and navigation equipment), high quality components were selected for servos, control arms, propulsion and power distribution systems. The Pixhawk autopilot is the latest advanced autopilot, also widely used by hobbyists, and has been utilized on the rovers before. Previous flight tests by the manufacturer have established a well-defined set of tuned gains, specific to the X8, that will be used with the X8. Lab and field testing has also verified the range and capability of the telemetry system. The safety pilot who will be flying the X8 has numerous hours flying remote controlled airframes.

c. Previous lessons learned:
The team has spent 2 days TDY to camp Atterbury and has seen and operated the air vehicles. Lessons learned include verifying software integration and radio communication before leaving for the field. Plan for interruptions in operations based on other users in the area. The X8 with Pixhawk autopilot and mission planner will be the ground control system interface with the air vehicles. All screws and motors will be verified before launch.

d. Authorized flight:
This flight is authorized by the AFIT MFR which was reviewed and approved by the Unmanned Aerial Systems Airworthiness office at AFLCMC.

SUAS Preflight Checklist

Checklist to be run before each UAV flight

Before commencing preflight, calculate the operational risk with the ORM Checklist Form

UAV Setup		**Check**
1. Assemble UAV	Make all wiring connections. Install propeller. Do not connect batteries yet.	
2. Inspect UAV	Check props and hub for damage or fatigue. Inspect flight control surfaces for damage. Tighten assembly as needed. Check Center of Gravity (CG) location.	
3. Install Fully Charged Battery	Connect battery cable.	
	Adjust batteries and/or weight/ballast position as necessary to ensure proper placement of CG.	
	Ensure batteries are properly secured.	

Autopilot Setup		
1. Prep Transmitter	Power on transmitter and set to "manual".	
2. Power On Autopilot Board	If no power switch is installed, you must disconnect then reconnect the battery.	
	Continue to keep the UAV level until the three colored LEDs stop flashing on the autopilot board (~30 sec).	
3. Obtain GPS Lock	Watch for the blinking red light on the APM to turn solid, indicating GPS lock. Can take up to 2 minutes.	

Communication/Ground Control Station		
1. Establish Communications	Follow Mission Planner procedures to ensure comms are established with APM. Perform comm. ground check to ensure proper range performance for autopilot comm. and RC receiver.	
2. Ensure Proper Gains Loaded	Check to make sure the correct gains are loaded for the UAV you are flying.	
3. Load Waypoints	Ensure proper waypoints are loaded and that a rally point (return to launch location) is loaded. RALLY POINT IS REQUIRED FOR LOST COMMUNICATIONS SCENARIO	

Take Off		
1. Obtain Clearance	Contact the field controller and obtain clearance to launch the UAV.	
2. Launch	Have assistant place aircraft at launch point.	
	ENSURE ALL PERSONNEL ARE CLEAR AND WARNED PRIOR TO ENGAGING PROPELLER.	
	Safety Pilot starts motor and executes a takeoff	

ORM Checklist Form

Date: _____ Control #:_____

	GREEN	YELLOW	RED
Crew Rest	Good	Marginal	Poor
Crew/Personal Concerns	None	Minor	Major
Primary Crew Qualified	All Qualified	1 Unqualified	2+ Unqualified
7+ Days TDY/Leave	2nd duty day back or later	1st duty day back	
Perceived Scheduling Pressure	None	Some	Significant Pressure to Complete Mission
Duty Day	<8 hours	>8 hrs	<12 hours
Showtime	0600-1600	0300-0600/1600-2200	2200-0300
Planning Changes (Last 24 hrs)	Minimal/No impact	Minor	Major
Mission Complexity	Low/Normal	Demanding	Extremely Demanding
Test Mission/Safety Risk	Low	Medium	High
Cross Winds/Wind Speed	<10 kts	10-13 kts	13-15 kts
Time of Day	Day	Night	0200-0500 TO/Landing
Airframe Modification	Minor	Significant	Severe
Maturity-Hardware/Software	Nothing New	1st Flight of Hardware/Software Mod	1st Flight of NEW Hardware/Software
Additional Risk Not Addressed	Low	Medium	High

This checklist is to be briefed at the beginning of each test day.

Each green box is 0 points. Each yellow box is 1 point. Each red box is 2 points.

- A score of 0-3: Attempt to mitigate any red boxes to reduce the risk. Test director's discretion to continue the mission.

- A score of 3-5: If unable to lower the score to 0-3, it is the Principal Investigator's discretion to continue the mission.

- A score of 6 or higher: If unable to lower the score, it is the AFIT FTSO's discretion as to whether or not to continue the mission.

IF YOU ARE NOT READY TO FLY… DON'T!

TEST DESCRIPTION

Objective 1 – Use DOE optimum parameter settings for reduced latency between 1 multi-rotor (Leader) and 1 rover (Follower)

<table>
<tr><th colspan="2">TEST SCENARIO 1</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings, found from Design of Experiments (DOE) with rovers, for the lowest latency between a multi-rotor (set as leader) and a rover (set as follower). Leader will be in manual. Follower will be controlled by Python script, but can still be switched into manual at any time.</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture time from when Leader movement starts until Follower vehicle responds</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
The follower vehicle responds as fast (~2 seconds) or faster to the leader vehicle as with a rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Fast user response on manual timing of vehicles
3. Functioning Stopwatch</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for a quick 2 second or better reaction time for the follower vehicle in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor
2. 2 GCS (laptops)
3. Stopwatch</td></tr>
<tr><td>Test Methodology</td><td>Test Procedures
1. BEFORE TAKEOFF:
a. Setup ground control station and operating area.
b. Load the appropriate parameter settings for the test point.
c. Preload Python Scripts on Leader GCS and Follower GCS.</td></tr>
</table>

	d. Complete SUAS preflight checklist. e. Check that weather is within limits and determine launch/recovery locations and headings. f. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position the aircraft for launch. d. Safety pilot executes takeoff. e. Safety pilot announces that air vehicle is airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCS. b. Manually move the leader multi-rotor. c. Start the timer. d. Wait for follower to move in response to the Leader. e. Stop timer once Follower responds by moving towards Leader. f. Record test point. g. Stop running Python scripts on Leader and Follower GCS. h. Transition vehicles to manual for recovery. 4. RECOVERY: a. Navigate aircraft to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilot announces landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop and ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. d. Power off ground control station as required.

Objective 2 – Use DOE optimum parameter settings for reduced accuracy error between 1 multi-rotor (Leader) and 1 rover (Follower)

<table>
<tr><th colspan="2">TEST SCENARIO 2</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings, found from Design of Experiments (DOE) with rovers, for the lowest accuracy error between a multi-rotor (set as leader) and a rover (set as follower). Accuracy error is determined by obtaining the average distance from the follower vehicle to the leader vehicle through the Telemetry Log (TLOG). Leader will be in Auto mode, following a predefined Figure 8 pattern through waypoints. The leader multi-rotor will be consistently held at a 10 m altitude (AGL). Follower will be controlled by Python script, but can still be switched into manual at any time.</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture accuracy error from TLOG after leader vehicle follows figure-8 pattern, with follower vehicle following leader vehicle</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
TLOGs and waypoint distance (WP Dist) of follower are recorded. Accuracy error obtained is comparable (~16 inches), if not better, than rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Recorded TLOG w/ WP Dist</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for an accuracy error of 16 inches or better for the follower vehicle in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor
2. 2 GCS (laptops)</td></tr>
</table>

Test Methodology	**Test Procedures** 1. BEFORE TAKEOFF: a. Setup ground control station and operating area. b. Load the waypoints onto the leader (multi-rotor) vehicle. c. Load the appropriate parameter settings for test point. d. Preload Python Scripts on Leader GCS and Follower GCS. e. Complete SUAS preflight checklist. f. Check that weather is within limits and determine launch/recovery. locations and headings. g. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with the air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position vehicles for launch. d. Safety pilot executes takeoff. e. Safety pilot announces that air vehicle is airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCS. b. Switch leader (multi-rotor) vehicle into Auto mode. c. Wait for leader to travel loop 5 times, with follower vehicle following d. Record test point. e. Stop running Python scripts on Leader and Follower GCS. f. Transition vehicles to manual for recovery. 4. RECOVERY: a. Navigate aircraft to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilot announces landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. a. Power off ground control station as required.

Objective 3 – Use DOE optimum parameter settings for reduced latency between 1 rover (Leader) and 1 multi-rotor (Follower)

<table>
<tr><th colspan="2">TEST SCENARIO 3</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings, found from Design of Experiments (DOE) with rovers, for the lowest latency between a rover (set as leader) and a multi-rotor (set as follower). Leader will be in manual. Follower will be controlled by Python script, but can still be switched into manual at any time. The follower multi-rotor will be consistently held at a 10 m altitude (AGL).</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture time from when Leader movement starts until Follower vehicle respond</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
The follower vehicle responds as fast (~2 seconds) or faster to the leader vehicle as with a rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Fast user response on manual timing of vehicles
3. Functioning Stopwatch</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for a quick 2 second or better reaction time for the follower vehicle in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor
2. 2 GCS (laptops)
3. Stopwatch</td></tr>
<tr><td>Test Methodology</td><td>Test Procedures
1. BEFORE TAKEOFF:
a. Setup ground control station and operating area.
b. Load appropriate parameter settings for test point.
c. Preload Python Scripts on Leader GCS and Follower GCS.
d. Complete SUAS preflight checklist.</td></tr>
</table>

	e. Check that weather is within limits and determine launch/recovery locations and headings. f. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with each air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position aircraft for launch. d. Safety pilot executes takeoff. e. Safety pilot announces that air vehicle is airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCS. b. Manual move the leader rover. c. Start timer. d. Wait for follower to move in response to Leader. e. Stop timer once Follower responds. f. Record test point. g. Stop running Python scripts on Leader and Follower GCS. h. Transition vehicles to manual for recovery. 4. RECOVERY: a. Navigate aircraft to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilot announces landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. d. Power off ground control station as required.

Objective 4 – Use DOE optimum parameter settings for reduced accuracy error between 1 rover (Leader) and 1 multi-rotor (Follower)

<table>
<tr><th colspan="2">TEST SCENARIO 4</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings, found from Design of Experiments (DOE) with rovers, for the lowest accuracy error between a rover (set as leader) and a multi-rotor (set as follower). Accuracy error is determined by obtaining the average distance from the follower vehicle to the leader vehicle through the Telemetry Log (TLOG). Leader will be in Auto mode, following a predefined Figure 8 pattern through waypoints. The leader multi-rotor will be consistently held at a 10 m altitude (AGL). Follower will be controlled by Python script, but can still be switched into manual at any time. The follower multi-rotor will be consistently held at a 10 m altitude (AGL).</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture accuracy error from TLOG after leader vehicle follows figure-8 pattern, with follower vehicle following leader vehicle</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
TLOGs and waypoint distance (WP Dist) of follower are recorded. Accuracy error obtained is comparable (~16 inches), if not better, than rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Recorded TLOG w/ WP Dist</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for an accuracy error of 16 inches or better for the follower vehicle in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor
2. 2 GCS (laptops)</td></tr>
</table>

Test Methodology	**Test Procedures** 1. BEFORE TAKEOFF: a. Setup ground control station and operating area. b. Load waypoints onto leader (rover) vehicle. c. Load appropriate parameter settings for test point. d. Preload Python Scripts on Leader GCS and Follower GCS. e. Complete SUAS preflight checklist. f. Check that weather is within limits and determine launch/recovery locations and headings. g. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with the air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position vehicles for launch. d. Safety pilot executes takeoff. e. Safety pilot announces that air vehicle is airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCS b. Switch leader (rover) vehicle into Auto mode c. Wait for leader to travel loop 5 times, with follower vehicle following d. Record test point. e. Stop running Python scripts on Leader and Follower GCS f. Transition vehicles to manual for recovery. 4. RECOVERY: a. Navigate aircraft to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilot announces landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. d. Power off ground control station as required.

Objective 5 – Use DOE optimum parameter settings for reduced latency between 2 multi-rotors (Leader/Follower)

<table>
<tr><th colspan="2">TEST SCENARIO 5</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings, found from Design of Experiments (DOE) with rovers, for the lowest latency between a leader and follower multi-rotor. Leader will be in manual. Follower will be controlled by Python script, but can still be switched into manual at any time. Altitude of leader will remain constant by manually controlling the vehicle at an altitude of 10m (AGL). The follower will be consistently set at 3 m higher than the leader to avoid collision.</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture time from when Leader movement starts until Follower vehicle respond</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
The follower vehicle responds as fast (~2 seconds) or faster to the leader vehicle as with a rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Fast user response on manual timing of vehicles
3. Functioning Stopwatch</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for a quick 2 second or better reaction time for the follower vehicle in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor
2. 2 GCS (laptops)
3. Stopwatch</td></tr>
<tr><td>Test Methodology</td><td>Test Procedures
1. BEFORE TAKEOFF:
a. Setup ground control station and operating area.</td></tr>
</table>

	b. Load appropriate parameter settings for test point. c. Preload Python Scripts on Leader GCS and Follower GCS. d. Complete SUAS preflight checklist. e. Check that weather is within limits and determine launch/recovery locations and headings. f. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with each air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position air vehicles for launch. d. Safety pilots execute takeoff. e. Safety pilots announce that air vehicles are airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCS. b. Manual move the leader air vehicle. c. Start timer. d. Wait for follower to move in response to Leader. e. Stop timer once Follower responds. f. Record test point. g. Stop running Python scripts on Leader and Follower GCS. h. Transition to manual flight for recovery. 4. RECOVERY: a. Navigate aircraft to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilots announce landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. d. Power off ground control station as required.

Objective 6 – Use DOE optimum parameter settings for reduced accuracy error between 2 multi-rotors (Leader/Follower)

<table>
<tr><th colspan="2">TEST SCENARIO 6</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings, found from Design of Experiments (DOE) with rovers, for the lowest accuracy error between a leader and follower multi-rotor. Accuracy error is determined by obtaining the average distance from the follower vehicle to the leader vehicle through the Telemetry Log (TLOG). Leader will be in Auto mode, following a predefined Figure 8 pattern through waypoints. Altitude of leader will remain constant by maintaining the vehicle at an altitude of 10m (AGL) through recorded waypoints. Follower will be controlled by Python script, but can still be switched into manual at any time. The follower will be consistently set at 3 m higher than the leader to avoid collision.</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture accuracy error from TLOG after leader vehicle follows figure-8 pattern, with follower vehicle following leader vehicle</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
TLOGs and waypoint distance (WP Dist) of follower are recorded. Accuracy error obtained is comparable (~16 inches), if not better, than rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Recorded TLOG w/ WP Dist</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for an accuracy error of 16 inches or better for the follower vehicle in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor</td></tr>
</table>

	2. 2 GCS (laptops)
Test Methodology	**Test Procedures** 1. BEFORE TAKEOFF: a. Setup ground control station and operating area. b. Load waypoints onto leader vehicle. c. Load appropriate parameter settings for test point. d. Preload Python Scripts on Leader GCS and Follower GCS. e. Complete SUAS preflight checklist. f. Check that weather is within limits and determine launch/recovery locations and headings. g. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with each air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position air vehicles for launch. d. Safety pilots execute takeoff. e. Safety pilots announce that air vehicles are airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCS. b. Switch leader air vehicle into Auto mode. c. Wait for leader to travel loop 5 times, with follower vehicle following. d. Record test point. e. Stop running Python scripts on Leader and Follower GCS. f. Transition to manual flight for recovery. 4. RECOVERY: a. Navigate air vehicles to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilots announce landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. d. Power off ground control station as required.

Objective 7 – Run optimum DOE parameter settings with 1 rover (leader) and 2 multi-rotors (followers) for reduced latency

<table>
<tr><th colspan="2">TEST SCENARIO 7</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings found from previous models for lowest latency. Leader will be in manual. Followers will be controlled by Python script, but can still be switched into manual at any time. There will be one safety pilot for each Follower. The altitude of the multi-rotors will remain at a constant altitude of 10 m and 13 m. The multi-rotors will be controlled by the script, following the leader at an appropriate V configuration offset with a staggered altitude to avoid collision.</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture time from when Leader movement starts until Follower vehicles respond</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
The follower vehicles responds as fast (~2 seconds) or faster to the leader vehicle as with a rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Fast user response on manual timing of vehicles
3. Functioning Stopwatch</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for a quick 2 second or better reaction time for the follower vehicles in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor
2. 3 GCS (laptops)
3. Stopwatch</td></tr>
<tr><td>Test Methodology</td><td>Test Procedures
1. BEFORE TAKEOFF:
a. Setup ground control station and operating area.</td></tr>
</table>

	b. Load appropriate parameter settings for test point. c. Preload Python Scripts on Leader GCS and Follower GCSs. d. Complete SUAS preflight checklist. e. Check that weather is within limits and determine launch/recovery locations and headings. f. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with each air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position air vehicles for launch. d. Safety pilots execute takeoff. e. Safety pilots announce that air vehicles are airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCSs. b. Manual move the leader air vehicle. c. Start timer. d. Wait for followers to move in response to Leader. e. Stop timer once Followers respond. f. Record test point. g. Stop running Python scripts on Leader and Follower GCSs. h. Transition to manual flight for recovery. 4. RECOVERY: a. Navigate aircraft to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilots announce landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. d. Power off ground control station as required.

Objective 8 – Run optimum DOE parameter settings with 1 rover (leader) and 2 multi-rotors (followers) for reduced accuracy error

<table>
<tr><th colspan="2">TEST SCENARIO 8</th></tr>
<tr><td>Description</td><td>Run optimum parameter settings found from previous models for accuracy error. Accuracy error is determined by obtaining the average distance from the follower vehicle to the leader vehicle through the Telemetry Log (TLOG). Leader will be in Auto mode, following a figure-8 pattern, but can be switched to Manual control at any time. There will be one safety pilot for each Follower. Followers will be controlled by Python script, but can still be switched into manual at any time. The altitude of the multi-rotors will remain at a constant altitude of 10 m and 13 m. The multi-rotors will be controlled by the script, following the leader at an appropriate V configuration offset with a staggered altitude to avoid collision.</td></tr>
<tr><td>Stakeholders</td><td>1st Lt Stefan Hardy</td></tr>
<tr><td>Success Criteria</td><td>Completion of the following test matrix:
<table>
<tr><th>Test Point</th><th>Description</th><th>WP Radius (m)/Cruise Speed (m/s)/Sleep Time (ms)/Max Window (ms)/Telemetry Rate</th></tr>
<tr><td>1.1</td><td>Capture accuracy error from TLOG after leader vehicle follows figure-8 pattern, with follower vehicles following leader vehicle</td><td>1/1/500/33/10</td></tr>
</table></td></tr>
<tr><td>Evaluation Criteria</td><td><u>Satisfactory if:</u>
TLOGs and waypoint distance (WP Dist) of followers are recorded. Accuracy error obtained is comparable (~16 inches), if not better, than rover-to-rover relationship.</td></tr>
<tr><td>Data Requirements</td><td>Required
1. Functioning Python Script
2. Recorded TLOG w/ WP Dist</td></tr>
<tr><td>Algorithms</td><td>N/A</td></tr>
<tr><td>Expected Results</td><td>The parameter settings allow for an accuracy error of 16 inches or better for the follower vehicles in response to the leader vehicle.</td></tr>
<tr><td>Assets</td><td>1. 3DR X8 Multi-Rotor</td></tr>
</table>

	2. 3 GCS (laptops)
Test Methodology	**Test Procedures** 1. BEFORE TAKEOFF: a. Setup ground control station and operating area. b. Load waypoints onto leader vehicle. c. Load appropriate parameter settings for test point. d. Preload Python Scripts on Leader GCS and Follower GCSs. e. Complete SUAS preflight checklist. f. Check that weather is within limits and determine launch/recovery locations and headings. g. Open airspace with range control. 2. LAUNCH: a. Ensure that all present personnel are aware of launch. b. Ensure, at a minimum, one assigned observer will assist safety pilot in maintaining visual contact with each air vehicle. Additional observers will assist in maintaining situational awareness around the airfield and flight operations area. c. Position air vehicles for launch. d. Safety pilots execute takeoff. e. Safety pilots announce that air vehicles are airborne. f. Climb to pre-briefed transition altitude. g. Transition to pre-briefed test-point entry position. 3. EXECUTE TEST POINTS: a. Start Python scripts on Leader GCS and Follower GCSs b. Switch leader vehicle into Auto mode c. Wait for leader to travel loop 5 times, with follower vehicles following d. Record test point. e. Stop running Python scripts on Leader and Follower GCSs f. Transition to manual flight for recovery. 4. RECOVERY: a. Navigate vehicles to pre-briefed recovery transition location. b. Ensure landing area is clear of personnel and equipment. c. Begin descent and entry into landing pattern. d. Safety pilots announce landing to all present personnel. e. Execute recovery. 5. AFTER RECOVERY: a. Stop telemetry capture on laptop or ensure that data log is saved. b. Close airspace with range control. c. Power off RC transmitter as required. e. Power off ground control station as required.

SAFETY PLAN

1. QUALIFICATION AND TRAINING
    a. Dr. David Jacques – Lead Faculty member of the AFIT UAS program. Experienced in UAS simulation and real world testing.
    b. Mr. Rick Patton – CESI employee and safety pilot with many years of experience flying R/C aircraft.
    c. 1st Lt Stefan Hardy – Successfully completed the Camp Atterbury Range Safety course and has his range control safety card.

2. GENERAL MINIMIZING CONDITIONS

    The following general minimizing procedures and considerations will be followed for the duration of this flight test program:
    1. All test flights will be conducted in day VMC conditions.
    2. A safety pilot will be used for all flights.
    3. Communications will be maintained between the ground operator, safety observers, sensor operator, and safety pilots at all times.
    4. The safety pilots will maintain positive radio communications with Himsel AAF Unicom at all times.
    5. Flying over non-participating personnel and facilities will be avoided.
    6. Personnel without assigned roles for a given test will be observers of flight operations while outside the flight test trailer. Minimize all unnecessary conversations and distractions during critical powered ground operations or flight.
    7. A multi-purpose fire extinguisher is readily accessible during all ground operations, especially during engine start-up.
    8. Utilize "Knock-It-Off" and "Terminate" procedures in accordance with AFI 11-214 paragraph 3.4.
    9. Maintain visual contact with aircraft at all times. If visual contact is lost, safety pilots initiate a "Return-to-Launch" via RC control.
    10. A safety Manual switch was programmed into the Python script so that the Safety Pilots' Manual control can override the script and function of the follower vehicle at any time deemed necessary or unsafe.

3. TEST HAZARD ANALYSES (THA's)
    A. Battery Fire
    B. Collision with Object
    C. Collision with Personnel
    D. Total Loss of Communication with the Vehicle
    E. Loss of Control
    F. GPS Signal Loss

Probability of Mishap Occurring During the Test		Mishap Severity Category			
		Catastrophic – I Death, System/Facility Loss, Severe Environmental Damage (e.g. Class A Mishap)	**Critical – II** Severe Injury, Occupational Illness, or Major System/Facility/ Environmental	**Marginal – III** Minor Injury, Occupational Illness, or Minor System/Facility/ Environmental	**Negligible – IV** Less than Minor Injury, Occupational Illness, or System/ Facility/ Environmental Damage
	Very Likely (A) Highly expected to occur – Many significant concerns even after mitigation applied.	**1**	**3**	**7**	**13**
	Likely (B) Expected to occur – Significant concerns remain after mitigation applied.	**2**	**5**	**9**	**16**
	Less Likely (C) Not expected but possible – Some concern exists even with mitigation applied.	**4**	**6**	**11**	**18**
	Unlikely (D) Unexpected – Minor concerns after mitigation applied.	**8**	**10**	**14**	**19**
	Very Unlikely (E) Highly unexpected – Little or no concern after mitigation applied.	**12**	**15**	**A, C, D, E** **17**	**B, F** **20**

HIGH

MEDIUM

LOW

NEGLIGIBLE

TEST HAZARD ANALYSIS (THA)	Page 1/6
TEST SERIES Lt Hardy Thesis Work (All Test Points)	MISHAP CAT/PROBABILITY III/Very Unlikely
PREPARED BY Stefan Hardy, 1st Lt, USAF	SIGNATURE
AFIT FLIGHT TEST SAFETY OFFICER Jeremy Agte, Lt Col, USAF	SIGNATURE

HAZARD: Battery Fire

CAUSE:

1. Uncontrolled discharge of power from the battery leading to overheating and fire (thermal runaway)
2. Overcharging of battery leading to thermal runaway due to charger malfunction or human input error
3. Battery circuitry or subsystem component failure or wiring malfunction

EFFECT:

1. Loss of vehicle
2. Injury to personnel

MINIMIZING PROCEDURES:

1. (1,2,3) All batteries will be stored in fireproof containers.
2. (1,2,3) All batteries will be charged using authorized battery chargers and by personnel trained in the proper recharging techniques.
3. (1,2,3) All batteries will be charged in AFIT approved locations.
4. (1,3) Only the proper battery types for the specified aircraft will be used (no smaller or larger capacity batteries used).
5. (1,2,3) Load balancer will be used when charging flight batteries.

CORRECTIVE ACTIONS:

If the battery catches fire during ground operations:

1. The pilot in command will power off the transmitter.
2. The person nearest to the fire extinguisher will use the fire extinguisher to put out the fire.
3. The person in communication with the field controller will notify the field controller of the emergency via the radio.

If the battery catches fire while in flight:

1. Announce battery fire.
2. The pilot in command will immediately land the aircraft (make attempt to land on hard surface).
3. If controls are not available, all personnel will maintain a visual of the aircraft and notify the field controller of the emergency.
4. The aircraft observer (or person nearest to the fire extinguisher) will use the fire extinguisher to put out the fire once the aircraft lands.
5. Follow mishap reporting procedures per section IV of this document.

REMARKS: None

TEST HAZARD ANALYSIS (THA)	Page 2/6
TEST SERIES Lt Hardy Thesis Work (All Test Points)	MISHAP CAT/PROBABILITY IV/ Unlikely
PREPARED BY Stefan Hardy, 1st Lt, USAF	SIGNATURE
AFIT FLIGHT TEST SAFETY OFFICER Jeremy Agte, Lt Col, USAF	SIGNATURE

HAZARD: Collision with Object

CAUSE:

1. Bird strike
2. Collision with other aircraft
3. Collision with ground based obstructions

EFFECT:

1. Loss of vehicle
2. Property damage

MINIMIZING PROCEDURES:

1. (1,2,3) Safety observers will be used to augment operator and safety pilot.
2. (2) Communicate with the tower before testing to verify clear airspace.
3. (3) Flight path will be adjusted in order to avoid ground based obstructions.

CORRECTIVE ACTIONS:

1. Announce collision with object.
2. Discontinue testing and verify there are no injuries.
3. Notify tower if hit or near miss with non-AFIT air vehicle occurs.

REMARKS:

1. Follow mishap reporting procedures per section IV of this document.
2. Document exact damage with photos/video.
3. Examine and repair vehicle if damaged.
4. When/if operational, perform a trim flight to ensure safe, stable flight and functionality.

TEST HAZARD ANALYSIS (THA)	Page 3/6
TEST SERIES Lt Hardy Thesis Work (All Test Points)	MISHAP CAT/PROBABILITY III/Very Unlikely
PREPARED BY Stefan Hardy, 1st Lt, USAF	SIGNATURE
AFIT FLIGHT TEST SAFETY OFFICER Jeremy Agte, Lt Col, USAF	SIGNATURE

HAZARD: Collision with Personnel

CAUSE:

1. Unexpected personnel interference during takeoff/landing
2. Loss of control of vehicle

EFFECT:

1. Personnel injury
2. Loss of vehicle

MINIMIZING PROCEDURES:

1. (1) Launch/landing area will be cleared of all nonessential personnel during these phases of flight and launch and recovery of the aircraft will be announced loudly to all personnel.
2. (1, 2) All personnel will maintain situational awareness of vehicle/flight status and personnel in and around the test area.

CORRECTIVE ACTIONS:

1. Discontinue testing and determine if there are injuries.
2. All emergency services will be coordinated through range control (812-526-1351) if severe; perform any necessary first aid until help arrives.

REMARKS:

1. Follow mishap reporting procedures per section IV of this document.
2. Examine and repair vehicle if damaged
3. When/if operational, perform a trim flight to ensure safe, stable flight and functionality

TEST HAZARD ANALYSIS (THA)	Page 4/6
TEST SERIES Lt Hardy Thesis Work (All Test Points)	MISHAP CAT/PROBABILITY III/Very Unlikely
PREPARED BY Stefan Hardy, 1st Lt, USAF	SIGNATURE
AFIT FLIGHT TEST SAFETY OFFICER Jeremy Agte, Lt Col, USAF	SIGNATURE

HAZARD: Total Loss of Communication with the Vehicle

CAUSE:

1. Outside signal interference
2. RC controller/comm box/receiver failure
3. GCS power failure
4. Vehicle out of range

EFFECT:

1. Vehicle falls to ground or flies to pre-programmed waypoint
2. Unplanned off-field landing
3. Loss of control of aircraft (early PID flights)

MINIMIZING PROCEDURES:

1. (1,2,3,4) Verify operation of communication equipment prior to test. Verify integrity of the antennae. Verify communication equipment batteries are fully charged.
2. (1) Coordinate flight operations and frequencies with Atterbury authorities.
3. (1,2,3,4) Lost link fail-safes will be pre-programmed.
4. (1,2,3,4) Pre-flight checklist will be conducted.
5. (2,3) Computers will have backup batteries as well as external UPS.

CORRECTIVE ACTIONS:

1. Pilot and/or safety pilot will immediately announce lost communications so the test team can help visually track the vehicle.
2. Attempt to re-establish communications while the vehicle executes its pre-programmed lost link procedures.
3. If link cannot be re-established, discontinue testing.
4. Notify Himsel AAF UNICOM of UAV status.
5. If unplanned landing occurs, verify there are no injuries.
6. Follow mishap reporting procedures per section IV of this document.

REMARKS: None

TEST HAZARD ANALYSIS (THA)	Page 5/6
TEST SERIES Lt Hardy Thesis Work (All Test Points)	MISHAP CAT/PROBABILITY III/Very Unlikely
PREPARED BY Stefan Hardy, 1st Lt, USAF	SIGNATURE
AFIT FLIGHT TEST SAFETY OFFICER Jeremy Agte, Lt Col, USAF	SIGNATURE

HAZARD: Loss of Control

CAUSE:

1. GCS power failure
2. Servo failure
3. Structural failure

EFFECT:

1. Damage to vehicle
2. Damage to property or injury to personnel
3. Loss of vehicle

MINIMIZING PROCEDURES:

1. (1,2) Bench-test flight configuration prior to test day.
2. (1) Back-up power supplies will be used.
3. (2, 3) Visual inspection of the air vehicle will be accomplished prior to flight.
4. (2,3) Perform preflight control check.

CORRECTIVE ACTIONS:

1. Announce loss of control.
2. If in auto mode, safety pilots take control of the UAV for emergency maneuvers.
3. Discontinue testing and verify there are no injuries or property damage.
4. Follow mishap reporting procedures per section IV of this document.
5. Examine and repair vehicle if damaged.
6. If operational, perform a trim flight to ensure functionality.

REMARKS: None

TEST HAZARD ANALYSIS (THA)	Page 6/6
TEST SERIES Lt Hardy Thesis Work (All Test Points)	MISHAP CAT/PROBABILITY IV/Very Unlikely
PREPARED BY Stefan Hardy, 1st Lt, USAF	SIGNATURE
AFIT FLIGHT TEST SAFETY OFFICER Jeremy Agte, Lt Col, USAF	SIGNATURE

HAZARD: GPS Signal Loss

CAUSE:

1. Signal interference
2. Receiver failure
3. Poor receiver/satellite geometry
4. Connector failure

EFFECT:

1. Loss of navigation (autopilot will not fly to waypoints)
2. Unplanned off-field landing

MINIMIZING PROCEDURES:

1. (1,2,3,4) Plan for manual control changeover in event of lost GPS.

CORRECTIVE ACTIONS:

1. Announce GPS loss.
2. Safety pilots maintain controlled flight.
3. If GPS is not re-acquired as determined by test team, recover the UAV using manual mode.

REMARKS: None

AFTER ACTION REPORT

1. Use this section to briefly describe how the test was carried out. Were there any unusual events?

First, waypoints were written for the rover for the accuracy figure eight loop tests. The map loaded from Mission of the UAS strip wasn't updated. Therefore, the waypoints were always a little off. However, they were adjusted properly.

For the rover following the multi-rotor vehicle configuration, the rover seemed to follow fine. However, for the multi-rotor following rover vehicle configuration, the multi-rotor would immediately try to land at home when the Python script began running. Troubleshooting took place soon after. A single multi-rotor was operated using Guided Mode's "Fly to Here" on Mission Planner. The multi-rotor operated successfully. Then the single multi-rotor was operated from a Python script with manual waypoints set in order to test if Python was a limitation. The multi-rotor followed the waypoint from the script effectively. It was originally thought that writing the waypoints after switching to Guided Mode in the Python script was the reason for the multi-rotor's return to home action. However, this was proven to not be a problem since this script was used in the single vehicle multi-rotor test.

Finally, it was found that the vehicles operated in a sequence. With the rover and multi-rotor each connected to their respective GCSs, Guided Mode's "Fly to Here" was repeated with the multi-rotor. This time the multi-rotor immediately tried to land at its home location, just like when the Python script was originally run with both vehicles. Therefore, the rover was re-connected while the multi-rotor remained on. The Guided Mode's "Fly to Here" was performed again while both vehicles were connected to their GCSs and the multi-rotor performed successfully. Then the script was run and both vehicles performed effectively. The multi-rotor successfully followed the rover. The original vehicle configuration connections involved connecting the rover first, before the multi-rotor, hence the errors experienced.

2. What test execution/safety lessons were learned during the test event?

Being knowledgeable about equipment is a priority before experimentation. There wasn't much experience with the multi-rotors and so the operation of the multi-rotors was being learned as experimentation took place. Therefore, errors occurred when integrating other vehicles to the configuration.

ACRONYMS

AAF - Army Airfield
APM—Ardupilot Mega
AFLCMC—Air Force Life Cycle Management Center
CG - Center of Gravity
DOE – Design Of Experiments
ESC—Electronic Speed Control
FTSO—Flight Test Safety Officer
GCS – Ground Control Station
GPS—Global Positioning System
HHA – Himsel Army Airfield
LOS—Line of Sight
MFR—Military Flight Release
RC—Radio Controlled
UAS—Unmanned Aerial System
UAV—Unmanned Aerial Vehicle
UNICOM—Universal Communications
VMC—Visual Meteorological Conditions

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
26-03-2015	Master's Thesis	Sep 2013 – Mar 2015

TITLE AND SUBTITLE

Implementing Cooperative Behavior & Control Using Open Source Technology Across Heterogeneous Vehicles

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Hardy, Stefan L, 1st Lt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENY)
2950 Hobson Way,
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-15-M-180

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory, Munitions Directorate
Attn: Kevin M. Brink
101 W. Eglin Blvd
Eglin AFB, FL 32542-6810
(850) 872-4600 kevin.brink@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RW

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

This thesis describes the research effort into implementing cooperative behavior and control across heterogeneous vehicles using low cost off-the-shelf technologies and open source software. Current cooperative behavior and control methods are explored and improved upon to build analysis models. These analysis models characterize ideal factor settings for implementation and establish limits of performance for these low cost approaches to cooperative behavior and control.

The research focused on latency and position accuracy as the two measures of performance. Three different ground control station (GCS) software applications and two types of vehicles, rover ground vehicles and aerial multi-rotors, were used in this research. Using optimum factor settings from Design of Experiments (DOE), the multi-rotor following rover vehicle configuration experienced almost twice the latency of other experiments but also the lowest positional error of 0.8 m. Results show that the achieved update frequency of 0.5 Hz or slower would be far too slow for close-formation flight.

15. SUBJECT TERMS
Cooperative Behavior & Control, Heterogeneous vehicles, SUAS, multi-rotor, rover ground vehicle, vehicle following, Design of Experiments, Python, Mission Planner, Swarming

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. David R. Jacques, AFIT/ENV
U	U	U	UU	194	19b. TELEPHONE NUMBER *(Include area code)* (937) 255-6565, x3329 david.jacques@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18